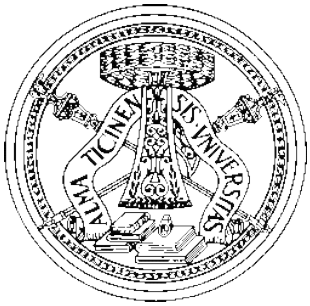


Università degli Studi di Pavia
Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

Sviluppo di una piattaforma distribuita per il monitoraggio del malware in Internet

Relatore:
prof. Giuseppe F. Rossi

Correlatore:
dott. Emanuele Goldoni

Tesi di laurea di:
Davide Cavalca
Matricola: 344501/56

Anno Accademico 2007/08

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Life is short and information endless: nobody has time for everything. In practice we are generally forced to choose between an unduly brief exposition and no exposition at all.

dalla prefazione di *Brave New World*
Aldous Huxley

La diffusa presenza di minacce alla sicurezza in Internet è oggi uno dei problemi principali della rete e dei suoi utenti. Negli ultimi anni, si è assistito ad un aumento esponenziale della diffusione e dell'ampiezza di queste minacce, accompagnato da un preoccupante mutamento negli scopi degli attaccanti. Storicamente, la fonte principale di problemi di sicurezza informatica era data dai *virus*: codice ostile che, tradizionalmente, colpiva "a pioggia", senza curarsi dell'obiettivo e spesso senza uno scopo ben definito, a di là della pura distruzione. Sempre più attacchi, invece, sono oggi progettati a tavolino per colpire direttamente un obiettivo predeterminato, con scopi che vanno dal furto di informazioni alla disabilitazione di un servizio. Gli attaccanti hanno scoperto una inesauribile fonte di guadagno nell'affitto di grandi reti di sistemi compromessi (*botnet*) ad organizzazioni criminali, che li utilizzano come piattaforme per l'invio di spam o per l'esecuzione di attacchi su larga scala, all'insaputa dei legittimi proprietari. Lo sviluppo di un vasto mercato sommerso di codice maligno, sistemi compromessi, nuove vulnerabilità e veri e propri servizi di consulenza nel crimine informatico, ha portato alla sempre maggior diffusione degli attacchi. Il risultato è che, oggi, anche un attaccante relativamente poco esperto dal punto di vista tecnico è in grado di sferrare attacchi potenzialmente devastanti, con conseguenze su larga scala. Questo cambio di prospettiva è illustrato chiaramente dal mutamento del linguaggio utilizzato: se in passato la minaccia principe era data dai virus, oggi si preferisce parlare più genericamente di *malware*, riferendosi al variegato insieme di tutti i programmi che possono avere un comportamento volutamente maligno.

Per poter affrontare al meglio le minacce presenti ed essere in grado di adottare tempestivamente le contromisure necessarie bisogna, prima di tutto, essere al corrente della loro esistenza e poterne monitorare lo stato. A questo scopo, solitamente vengono dispiegati dei sistemi trappola (*honeypot*, raccolti in reti dette *honeynet*), che espongono volutamente delle vulnerabilità note, con lo scopo di provocare degli

attacchi per poterne studiare la natura e la provenienza. Ad oggi le honeypot sono uno degli strumenti più utilizzati per la scoperta e lo studio delle minacce in rete. Permettono infatti di scoprire da un lato nuove tecniche d'attacco e vulnerabilità sfruttate da avversari ostili, dall'altro di acquisire il prima possibile campioni di nuovi malware auto-replicanti diffusi in rete.

Le honeynet sono ormai diventate uno strumento essenziale per ricercatori e operatori di rete. Ciononostante, la mancanza di un modello dei dati unificato le rende inevitabilmente meno efficaci, portando alla creazione di molteplici sorgenti di dati scorrelate tra loro, ognuna con il suo formato dei dati specifico e proprietario. Inoltre, il deployment e la gestione di una honeynet è un'attività impegnativa in termini di tempo e l'interpretazione dei dati raccolti è tutt'altro che banale. In questo lavoro verrà proposta HIVE (Honeynet Infrastructure in Virtualized Environment), una nuova infrastruttura di raccolta ed analisi, altamente scalabile e completamente automatizzata; HIVE è costruita sulle fondamenta di comprovate soluzioni Open Source, integrate con nuovi strumenti sviluppati *ad hoc*. Per semplificare la gestione ed il deployment delle honeypot viene fatto largo uso di tecnologie di virtualizzazione, combinando sensori di diverso tipo in un'infrastruttura comune. L'uso di un DBMS relazionale risponde alla necessità di eseguire un'analisi dettagliata dei dati per avere una veloce comprensione delle minacce; il database, inoltre, fornisce accesso e memorizzazione centralizzate per i dati raccolti, assicurandone la consistenza e l'integrità. Verranno anche presentate alcune tecniche per il monitoraggio attivo delle botnet centralizzate che sono state integrate in HIVE. Il lavoro è suddiviso come segue:

- il **Capitolo 2** esamina in dettaglio cosa si intende per *malware* e come sono strutturate le *botnet*;
- nel **Capitolo 3** sono presentate le *honeypot*, la principale tecnologia oggi utilizzata per la cattura e lo studio del malware, precisando le loro applicazioni e le criticità da considerare;
- il **Capitolo 4** introduce HIVE, l'infrastruttura di analisi realizzata in questo lavoro, descrivendo e motivando le scelte implementative e tecnologiche adottate;
- nel **Capitolo 5**, infine, vengono esaminati i dati raccolti per cercare di caratterizzare meglio il fenomeno allo studio.

È stata posta particolare cura nel garantire la completa riproducibilità degli esperimenti. A tale proposito, l'infrastruttura costruita è dettagliatamente documentata ed utilizza esclusivamente strumenti Open Source. Il codice di programmi e script realizzati per questo lavoro è pubblicato nelle Appendici A e B, sempre sotto licenza Open Source, nella speranza che possa essere utile per studi futuri.

Capitolo 2

Malware

equo ne credite, Teucri.
quidquid id est, timeo Danaos et dona ferentes.

Eneide, II.48–49
Virgilio

Con il termine *malware* si intende genericamente qualsiasi programma scritto con lo scopo di infiltrarsi in un computer senza il consenso del suo utilizzatore, danneggiandolo in vario modo. Il termine comprende i classici *virus* informatici, ma si estende anche a tutti quei programmi, che si sono diffusi a macchia d'olio negli ultimi anni, scritti con lo scopo di carpire dati personali, diffondere pubblicità o rendere controllabile da remoto un sistema. Si tratta quindi di una categoria molto ampia, che abbraccia sostanzialmente tutto il software "indesiderabile" che è possibile trovare su sistema. Nella Sezione 2.1 verrà presentata una breve tassonomia del genere, per comprendere meglio le diverse tipologie di malware presenti in rete, le loro principali caratteristiche e le minacce presentate.

Questo lavoro si concentra su una particolare categoria di malware, che si infiltra senza farsi notare in un sistema e lo collega ad una *botnet*, per renderlo controllabile da remoto da parte di un attaccante. Come mostrato in Figura 2.1, negli ultimi due anni c'è stata una crescita del 1500% nella quantità di minacce rilevate in rete; un contributo significativo è dato dall'espansione delle *botnet*.

2.1 Tipi di malware

La classificazione dei diversi tipi di malware non è semplice, a causa del gran numero di varianti diffuse; ciononostante, si ritiene che un raggruppamento per obiettivi possa facilitare la comprensione del fenomeno. Un primo gruppo è composto da tutti i programmi scritti con il solo scopo di causare dei danni e diffondersi il più possibile. Queste forme di vandalismo elettronico possono attaccare i singoli sistemi (i classici *computer virus*) o le infrastrutture di comunicazione (si parla in questo caso di *network worm*). Virus e worm sono in grado di riprodursi autonomamente (es. modificando il codice eseguibile dei programmi su disco); questa è una caratteristica comune a gran parte del malware. Il primo worm ad aver acquisito una certa diffusione è stato il Morris (così chiamato dal nome del suo autore): sfruttando un insieme di

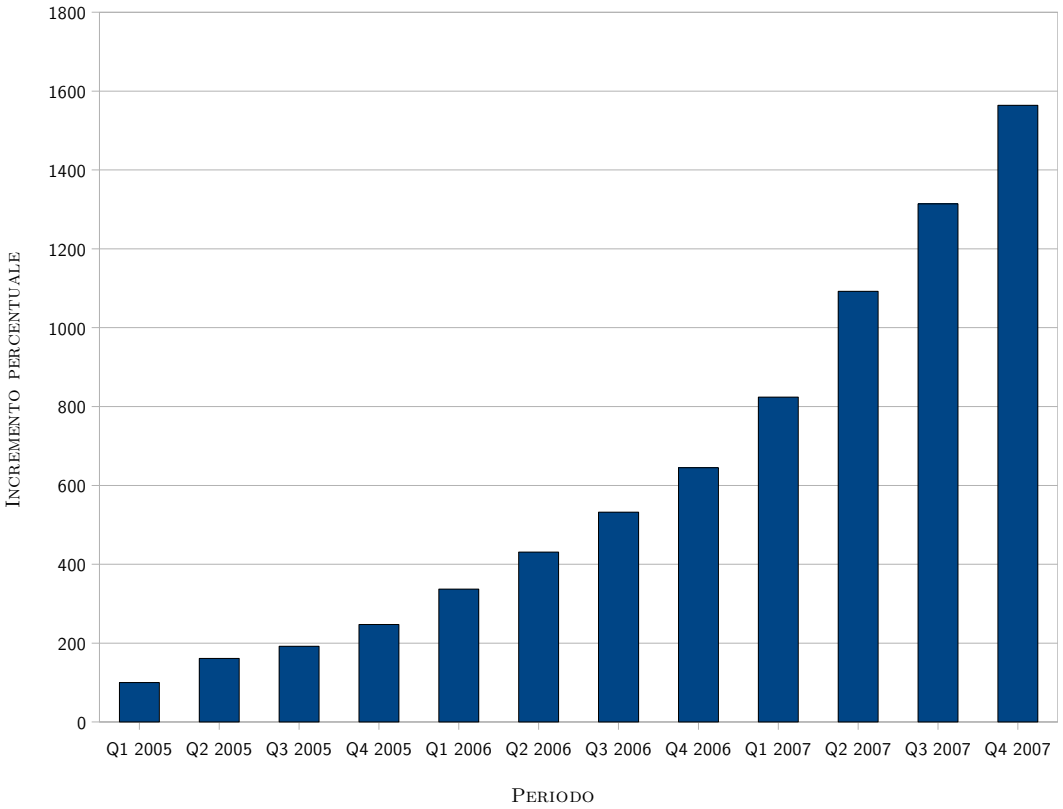


Figura 2.1: Aumento percentuale della diffusione del malware dal 2005 al 2007. Il grafico è tratto da [Trend Micro, 2007].

vulnerabilità software, era in grado di riprodursi su gran parte dei sistemi UNIX che all'epoca componevano Internet. Questo worm, che ha causato danni per decine di migliaia di dollari, è stato analizzato in dettaglio in [Eichin e Rochlis, 1989].

Nella stessa famiglia dei virus si possono collocare anche i *trojan horse*. Si tratta di programmi dall'apparenza innocua che nascondono però una componente distruttiva[1] (a volta chiamata *bomba logica*); a differenza della maggior parte dei malware, i trojan solitamente non si riproducono in modo automatico, ma vengono diffusi manualmente, spesso sfruttando tecniche di *social engineering*[2].

I tipi di malware esaminati finora non producono, generalmente, un vantaggio diretto per il loro creatore; si tratta sostanzialmente di programmi con finalità distruttive, più o meno intenzionali. Una seconda categoria invece, che è stata caratterizzata da una vera e propria esplosione nell'ultimo decennio, è quella dei malware *for profit*, scritti per ottenere un guadagno finanziario, generalmente illecito. Fanno parte di questa famiglia gli *adware*, programmi che diffondono pubblicità indesiderata, spesso in modo particolarmente invadente; una variante, estremamente diffusa, è data dagli *spyware*, che possono tracciare le abitudini di utilizzo e di navigazione web dell'utente. In questo caso il guadagno è dato dalla vendita delle informazioni acquisite o, più spesso, dalla semplice diffusione della pubblicità, che può essere più o meno mirata all'utente specifico. Si noti che adware e spyware non sono necessariamente illeciti di per sé: la licenza d'uso di un programma potrebbe prevedere l'installazione di componenti che, sebbene non siano esplicitamente identificati come tali, si comportano come spyware o adware; in tal caso sta alla responsabilità dell'utente evitarne l'installazione.

Altri esempi di malware sono dati dai *dialer*, che eseguono telefonate indesiderate a numeri a tariffazione aggiunta e dai *keylogger*, che registrano una copia di tutta l'attività dell'utente, compresi i tasti premuti, e possono essere usati per carpire password o altre informazioni riservate. I *ransomware* (o *cryptovirus*), invece, sono virus che cifrano i dati memorizzati sul sistema; l'obiettivo è in questo caso il ricatto dell'utente, che viene indotto a pagare una somma di denaro per ottenere la chiave di decodifica (senza ovviamente alcuna certezza di riceverla realmente). Un esempio di ransomware è dato da Gpcode [Kaspersky Lab, 2008], che cifra i file dell'utente usando l'algoritmo RSA per poi ricattarlo.

L'ultimo gruppo, infine, è dato dai programmi che rendono il sistema parte di una *botnet* per poterlo controllare da remoto. Questa categoria sarà esaminata in dettaglio nella Sezione 2.3.2.

2.2 Vettori d'attacco

Per vettore d'attacco si intende la modalità con cui il malware riesce ad insediarsi nel sistema. In molti casi si tratta di un fattore umano e non tecnologico: quando

[1] Il nome è un evidente riferimento al cavallo che, nella narrazione di [Virgilio, 1969], i greci costruiscono come stratagemma per invadere la città di Troia.

[2] Per *social engineering* si intende la capacità di manipolare le persone inducendole ad eseguire delle azioni o a rivelare informazioni riservate. È una tecnica simile a quelle utilizzate dai comuni truffatori per perpetrare delle frodi, ma non è quasi mai eseguita faccia a faccia ed ha lo scopo specifico di ottenere illecitamente accesso a dei sistemi informativi o acquisire informazioni riservate. Per maggiori informazioni fare riferimento a [Granger, 2001]

un utente riceve un'email con allegato un archivio protetto da password, non si può fare molto per proteggerlo se il testo del messaggio lo convince ad aprire l'archivio e lanciare il programma contenuto. Poiché sulla maggior parte dei sistemi in circolazione l'utente esegue i programmi con i massimi privilegi, il malware ottiene il controllo della macchina. Per quanto possa sembrare improbabile, questa ed altre tecniche di social engineering sono tra le modalità di attacco più diffuse. Oltre all'email, i malware sono diffusi attraverso sistemi di messaggistica istantanea e chiavette USB[3].

Se ci si limita a considerare i vettori tecnici, sicuramente il principale è la presenza di bachi nel software in esecuzione. Specialmente per quanto riguarda i network worm, la porta d'accesso verso il sistema è in genere un servizio vulnerabile o un problema nello stack di rete del sistema operativo. L'Internet Worm di Morris [Eichin e Rochlis, 1989], Code Red [Moore e Shannon, 2001] e Nimda [F-Secure, 2001b] si sono tutti diffusi sfruttando dei bachi nel software di rete. Un altro vettore di trasporto di malware sono i documenti che supportano un qualche tipo di linguaggio di macro eseguibile; l'esempio più lampante è Microsoft Word, che in passato eseguiva automaticamente all'apertura le macro contenute in un documento senza alcun controllo. Melissa [F-Secure, 2001a], uno dei più diffusi macro virus, colpiva proprio i documenti Word e sfruttava la rubrica di Outlook per replicarsi, inviando allegati infetti via email.

2.3 Le botnet

In questa sezione verrà esaminato un particolare tipo di malware, che una volta infettato il sistema resta nascosto e lo rende controllabile da remoto da un attaccante. Il sistema colpito diventa uno *zombie* ed entra a far parte di una *botnet*, una rete distribuita di programmi autonomi. Inviando comandi alla botnet (solitamente attraverso un *centro di controllo*), l'attaccante (*botherder*) è in grado di controllare tutti i sistemi infettati; potrebbe, ad esempio, ordinare alle macchine di eseguire un attacco DoS, espandersi cercando altri sistemi vulnerabili, inviare dello spam o aggiornare il malware installato.

Un malware di questo tipo deve essere il più possibile invisibile: spesso l'unico comportamento osservabile dall'utente è l'aumento di traffico di rete generato dal computer, o un generale calo di prestazioni. In alcuni casi, il collegamento ad una botnet è veicolato insieme ad uno spyware o un adware. Uno studio di alcuni software comunemente utilizzati per costruire una botnet è disponibile in [Barford e Yegneswaran, 2006]. In rete sono facilmente reperibili diversi pacchetti "preconfezionati" che permettono di costruire un malware in pochi minuti; se diffuso correttamente, un programma di questo tipo può portare alla creazione di una botnet in un tempo piuttosto breve.

[3] Anche in informatica, la storia tende a ripetersi. I primi computer virus si replicavano via floppy; il passaggio di dischetti non controllati tra amici era probabilmente il vettore principale di infezione. Con l'avvento dell'uso di massa di Internet e la progressiva scomparsa dei lettori di floppy dai PC, questa minaccia sembrava estinta. Nella realtà, il mezzo di diffusione sono ora diventate le chiavette USB che, al contrario dei floppy, possono contenere diversi gigabyte di dati e sono quindi molto più lente da controllare e ripulire.

2.3.1 Evoluzione della specie

Le tecniche di replicazione usate da questo tipo di malware sono certamente ispirate a quelle dei network worm: entrambi sfruttano le vulnerabilità e i bachi di sicurezza del software come veicolo primario d'infezione. Vi è però un elemento di novità, dato dall'uso sempre più frequente di tecniche di social engineering, verso le quali è molto difficile improntare una difesa.

Le prime botnet utilizzavano il protocollo IRC[4] per collegarsi ad un centro di controllo (C&C, dall'inglese *command and control*), dove l'attaccante monitorava il loro stato ed inviava i comandi di gestione. Questa tecnica è poco robusta: per distruggere la botnet è sufficiente abbattere il centro di controllo, che costituisce quindi un *single point of failure* della rete. Inoltre, poiché i comandi IRC viaggiano in chiaro, sono relativamente semplici da individuare, anche se non è banale distinguere una normale conversazione IRC dal traffico prodotto da un bot. Per rendere più difficoltoso il collegamento al centro di controllo da parte di utenti non autorizzati, molti attaccanti usano server IRC modificati, che non accettano i comandi standard o richiedono una fase preventiva di autenticazione. Ciononostante, monitorando il traffico generato dagli zombie è piuttosto facile scoprire la corretta sequenza di comandi. L'uso della cifratura renderebbe un'analisi di questo genere molto più ardua, ma ad oggi sostanzialmente tutti i bot di questo genere sono piuttosto semplici e comunicano in chiaro.

Una variante, che prevede sempre l'uso di un C&C centralizzato, utilizza il protocollo HTTP. In questo caso gli zombie richiedono periodicamente un documento HTTP (in genere con una GET), che l'attaccante aggiorna con i comandi da eseguire. Il controllo è in genere realizzato con soluzioni ad hoc (es. uno script PHP); le informazioni ricevute dallo zombie vanno da un URL da cui scaricare un aggiornamento del malware, ad un blocco di testo cifrato che il bot decodifica ed interpreta. Alcuni bot inviano essi stessi delle informazioni al C&C sotto forma di parametri delle query HTTP. Nonostante sia possibile utilizzare in generale qualsiasi protocollo per gestire la comunicazione, ad oggi la stragrande maggioranza di botnet centralizzate si appoggia ad IRC o HTTP, eventualmente con lievi modifiche.

Per ovviare ai problemi connessi con l'uso di un solo centro di controllo, alcuni bot permettono di eleggere uno zombie a C&C temporaneo: nel momento in cui il nodo smette di essere accessibile, la carica viene trasferita ad un altro sistema. L'uso di questa tecnica risolve in parte i problemi di affidabilità, anche se lo spostamento del C&C richiede il continuo aggiornamento di tutti i bot ed è quindi un'operazione piuttosto onerosa. Inoltre, non tutti gli zombie possono essere dei buoni centri di controllo: perché la botnet funzioni bene, è necessario che il C&C disponga di molta banda; per questo compito sono in genere preferiti sistemi ospitati su reti universitarie o di ricerca (caratterizzate, nel caso di istituzioni statunitensi, da domini `.edu`), che rispetto agli altri hanno solitamente una maggiore disponibilità di banda e tendono ad essere poco controllati. A questo proposito, tendono ad essere sempre più utilizzate reti *fast-flux*[5], che permettono di cambiare in modo estremamente

[4] IRC (*Internet Relay Chat*) è un protocollo per la gestione di conversazioni testuali (*chat*); è ottimizzato per le conversazioni di gruppo (molti-molti) ed è largamente utilizzato fin dagli albori di Internet.

[5] Una rete *fast-flux* è una rete di sistemi compromessi con associati dei record DNS pubblici in continuo cambiamento, anche nell'ordine di pochi minuti. Sono generalmente costruite attraverso

Risorsa	*Prezzo (USD)*
Singola installazione di un adware	0,30–0,02 $ in base alla posizione geografica
Pacchetto di malware, versione di base	1 000–2 000 $
Pacchetto di malware, con servizi aggiuntivi	prezzo variabile a partire da 20 $
Affitto di un kit di exploit per 1 ora	0,99–1 $
Affitto di un kit di exploit per 2,5 ore	1,60–2 $
Affitto di un kit di exploit per 5 ore	4 $ (può variare)
Copia non rilevata di un troiano per acquisire informazioni	80 $ (può variare)
Attacco DoS distribuito	100 $ al giorno
10 000 sistemi compromessi	1 000 $
Credenziali bancarie rubate	prezzo variabile a partire da 50 $
Un milione di indirizzi email appena raccolti	da 8 $ in su, in base alla qualità

Tabella 2.1: Esempio di mercato di scambio nell'economia sotterranea del 2007; dati tratti da [Trend Micro, 2007].

rapido la posizione di un centro di controllo sfruttando in modo opportuno i servizi di DNS dinamico; queste reti sono presentate in [The Honeynet project, 2007].

L'ultima evoluzione nella struttura delle botnet è data dall'uso di protocolli peer-to-peer (P2P) completamente decentralizzati. Come mostrato in [Grizzard e altri, 2007], i bot più recenti sono molto più difficili da studiare ed infiltrare, poiché non è più presente un centro di controllo e le comunicazioni sono spesso cifrate. Ciononostante, alcune botnet P2P sono state violate e smantellate con successo; si veda [Holz e altri, 2008] e [Amini, 2008] per due esempi di analisi a riguardo, relative alle botnet *Storm* e *Kraken*.

2.3.2 Utilizzi tipici

Una volta creata, una botnet costituisce una fonte di guadagno per il suo creatore, che può affittare[6] la banda o la potenza di calcolo degli zombie per i compiti più disparati. Gli utilizzi principali, studiati anche da [Govil e Govil, 2007], sono:

- Denial of Service: l'attaccante richiede a tutti gli zombie di eseguire connessioni verso un sistema, con lo scopo di sovraccaricarlo e renderlo inaccessibile;

- attacchi a reti IRC: l'attaccante ordina ai bot di collegarsi ad un canale IRC, che viene subissato di richieste e scollega tutti gli utenti; è una forma di DoS, ma in certi casi può consentire all'attaccante di ottenere il controllo del canale;

opportuni Resource Record (RR) DNS, frequentemente modificati, che combinano un gran numero di indirizzi IP di risoluzione con un TTL molto basso.

[6] È stata comprovata la presenza di un florido mercato sotterraneo di scambio, come studiato in dettaglio da [Franklin e altri, 2007]. Un esempio di "prezzario" tratto da [Trend Micro, 2007] è mostrato in Tabella 2.1.

- invio di spam: ogni zombie inizia a spedire della posta indesiderata ad una lista di indirizzi fornita[7];
- data mining: su ogni zombie è eseguita una ricerca per dati di interesse (indirizzi email, documenti, chiavi di registrazioni di giochi o software, ecc.), che sono poi inviati all'attaccante; la ricerca può essere eseguita sui dischi del sistema infettato, ma può anche fare uso di un keylogger o di uno sniffer di rete;
- diffusione di altro malware: lo zombie invia via email o utilizzando un servizio di messaggistica istantanea copie di malware agli indirizzi forniti (che possono eventualmente essere cercati sulla macchina infettata);
- frode pubblicitaria (*click fraud*): l'attaccante crea una pagina web con dei banner pubblicitari che pagano una certa somma per ogni click e ordina poi agli zombie di collegarsi al sito e scaricare la pubblicità; per massimizzare l'effetto, il browser dei sistemi zombie viene modificato per collegarsi frequentemente alle pagine pubblicitarie (ad esempio impostandole come homepage), in modo da aumentare il guadagno;
- falsare giochi o votazioni online: in modo analogo alla frode pubblicitaria, gli zombie si collegano al sito target e lasciano il loro voto; poiché ogni zombie ha un indirizzo IP diverso, appaiono tutti come persone differenti.

Un trend emerso relativamente di recente è l'uso di botnet per coadiuvare attività di *phishing*[8]. In questo caso l'attaccante, combinando diverse delle tecniche enunciate, usa gli zombie per ricercare potenziali vittime, inviare i messaggi ed ospitare i siti fasulli. Le credenziali ed i dati personali estratti dagli zombie permettono anche di eseguire agevolmente dei furti d'identità, qualora fosse necessario, ad esempio, registrare un dominio Internet o acquistare un certificato SSL.

2.3.3 Analisi e contromisure

Le tecniche per l'acquisizione di campioni di malware sono l'argomento del capitolo successivo; in questa sezione, invece, vengono esaminate brevemente le possibili azioni da adottare una volta individuata una botnet. Per impedire la comunicazione degli zombie con il centro di controllo, si può cercare di disturbare o bloccare il canale di comunicazione; se è stata individuata, ad esempio, una botnet IRC, il blocco del protocollo IRC a livello perimetrale dovrebbe disabilitare le comunicazioni. Nonostante appaia ragionevolmente semplice, una soluzione del genere si scontra

[7] Sempre più spesso i server di posta rifiutano i messaggi che originano da un indirizzo IP dinamico, poiché nella stragrande maggioranza dei casi si tratta proprio di messaggi prodotti da un bot. Alcuni ISP hanno anche iniziato a bloccare le connessioni in ingresso verso la porta 25, per impedire l'utilizzo di un sistema su IP dinamico come server di posta.

[8] Per *phishing* si intende il tentativo di indurre un utente a divulgare dati personali o finanziari per mezzo di messaggi di posta elettronica che appaiono provenire da una fonte fidata. Tipicamente, l'attaccante invia un messaggio, che rassomiglia quello di una nota istituzione (es. una banca), in cui spinge l'utente a seguire un link ed inserire delle informazioni (es. i codici di accesso all'home banking). Il link, in realtà, porta ad un sito, che generalmente imita anch'esso lo stile dell'istituzione impersonata, sotto il controllo dell'attaccante, che in questo modo è in grado di ottenere le informazioni inserite dall'utente. L'obiettivo del phishing è di solito il furto d'identità; per maggiori informazioni si veda [Ollmann, 2006]

con diverse difficoltà implementative. In primo luogo, non tutte le comunicazioni IRC sono illecite: un utente potrebbe usare IRC, ad esempio, per ricevere supporto tecnico online o per mantenersi in contatto con gli amici. Inoltre, assumendo anche che sia possibile bloccare tutto il traffico IRC, la sua identificazione è tutt'altro che semplice. Scartando a priori il filtraggio basato su coppie indirizzo-porta (poiché ogni malware usa una combinazione univoca e la cambia frequentemente), non resta che l'analisi dei pacchetti a livello applicativo (si parla di *layer7 filtering*); si tratta di una soluzione onerosa in termini di potenza di calcolo richiesta e non è necessariamente efficace, soprattutto in presenza di modifiche non documentate al protocollo di comunicazione. Nel caso di HTTP è di fatto impossibile attuare un filtraggio, a causa dell'uso massiccio che ne viene fatto per la normale navigazione web.

Come si è visto, è piuttosto difficile bloccare le comunicazioni di un bot; ciononostante, è possibile cercare di disturbarle o alterarle in qualche modo. Dei ricercatori sono riusciti, ad esempio, a decifrare il protocollo usato dalla botnet P2P *Kraken*; in [Amini, 2008] viene proposto un problema interessate: cosa fare se si ottiene la possibilità di iniettare comandi arbitrari negli zombie, magari forzando una disinstallazione del malware? Questa decisione, sebbene sia tecnicamente realizzabile, si scontra con problematiche di carattere etico e legale non banali. Se, infatti, uno dei sistemi infetti subisse un danno in seguito all'aggiornamento, i ricercatori ne sarebbero probabilmente ritenuti responsabili, trattandosi a tutti gli effetti di un accesso informatico non autorizzato, anche se eseguito "a fin di bene". Supponendo che uno dei sistemi infetti controlli un meccanismo di supporto vitale (es. un macchinario ospedaliero) o una centrale elettrica, il suo blocco potrebbe avere conseguenze critiche e del tutto imprevedibili.

Capitolo 3

Honeypot

You can't defend. You can't prevent. The only thing you can do is detect and respond.

Bruce Schneier

Lo stato delle minacce presenti in rete è in continua evoluzione: nuovi tipi di malware vengono diffusi ogni giorno, e nuove vulnerabilità nei programmi sono scoperte di frequente. Il modo più semplice per mantenersi al corrente delle minacce diffuse in rete in un dato periodo di tempo è ottenere una copia degli eseguibili del malware e studiarli in un ambiente controllato. In questo modo si possono ottenere informazioni su *cosa* effettivamente fa il malware, cercare di scoprirne le origini e lo scopo e implementare contromisure adeguate.

La cattura dei campioni è, sempre più spesso, effettuata per mezzo di *honeypot*, trappole virtuali studiate per attirare e catturare il malware, senza permettergli di causare danni. Le honeypot sono anche usate per attirare attaccanti umani e studiarne il comportamento. In questo capitolo vengono presentati i principali tipi di honeypot utilizzati, le loro caratteristiche e le loro limitazioni.

3.1 Introduzione

Una *honeypot*[1] è un sistema di rete che espone volutamente dei servizi vulnerabili, progettato per essere attaccato e compromesso senza causare danni a sistemi esterni e senza rischio di compromettere dati di una qualche importanza. Una honeypot non deve essere facilmente identificabile come tale: l'attaccante (che, come è già stato detto, può essere un umano o un malware auto-replicante) deve pensare che il sistema sia genuino e valga la pena di essere compromesso. Spesso, specie se l'obiettivo è attirare un attaccante umano e non un programma, è necessario costruire un insieme di dati fittizi (una vera e propria *esca*) per rendere più appetibile il sistema.

Un insieme di honeypot distribuite su una rete viene a creare una *honeynet*;

[1] Il termine *honeypot* fa riferimento al barattolo di miele che attira in modo irresistibile gli orsi nei parchi naturali; si tratta, probabilmente, di un riferimento scherzoso ai cartoni animati di Winnie-the-Pooh.

incrociando i dati raccolti da diverse honeynet distribuite geograficamente[2] si dovrebbe riuscire ad ottenere un quadro dello stato delle minacce a livello mondiale. Lo spazio di indirizzamento occupato da una honeynet è detto *darknet*; perché i dati non siano falsati, deve trattarsi di indirizzi inutilizzati da tempo e che non ospitano servizi legittimi.

3.2 Tipi di honeypot

Nell'ambito in esame come in molti altri, obiettivi diversi portano allo sviluppo ed all'adozione di soluzioni differenti. L'approccio utilizzato per studiare un attaccante umano è infatti diverso da quello necessario per raccogliere in modo automatizzato dei campioni di malware auto-replicante, nonostante entrambi facciano uso di honeypot. Inoltre, alcune applicazioni specifiche richiedono tecniche *ad hoc*, ispirate alle honeypot ma sostanzialmente differenti. In questa sezione vengono presentate le diverse categorie di honeypot sviluppate nel tempo, caratterizzando le applicazioni tipiche e le loro criticità.

3.2.1 Honeypot a bassa interazione

Una honeypot a bassa interazione (dall' inglese *low-interaction honeypot*) simula via software un'infrastruttura di rete con delle vulnerabilità note. A seconda dello scopo da raggiungere, la simulazione può riguardare un solo sistema o un'intera rete, ed il livello di dettaglio può essere più o meno accurato. Queste honeypot sono relativamente facili da rilevare: per quanto accurata, la simulazione non è mai perfetta ed è improbabile che copra tutti i possibili comportamenti dell'attaccante, specialmente se si tratta di un umano. Per contro, honeypot a bassa interazione sono poco costose da installare e da mantenere[3]. Le implementazioni sono relativamente poco onerose, ed è possibile simulare un'intera rete con centinaia di servizi attivi su una singola macchina. Inoltre, l'uso di una honeypot a bassa interazione comporta un rischio relativamente basso: poiché il codice del malware non viene effettivamente eseguito, a meno di bachi nell'implementazione della honeypot il sistema ospite è sicuro.

Sono presenti sul mercato diversi software per implementare una honeypot a bassa interazione. Uno dei primi programmi Open Source realizzati è stato *honeyd*, introdotto in [Provos, 2004]. Questo programma permette di simulare con una singola istanza un'intera rete personalizzando i servizi abilitati per ogni macchina; è principalmente orientato verso l'analisi di attaccanti umani. Nel corso degli anni sono stati sviluppati diversi software finalizzati alla cattura di campioni di malware auto-replicante. Il progetto più consolidato è sicuramente *Nepenthes*, descritto in [Baecher e altri, 2006]; tra gli altri, è utilizzato in diversi progetti di ricerca ed è il componente principale della rete di sensori della mwcollect Alliance [The mwcollect

[2] Ci riferiamo qui sia alla geografia fisica sia allo spazio di indirizzamento: le honeynet devono sì essere idealmente distribuite in modo uniforme in tutto il mondo, ma devono anche occupare equamente tutte le porzioni di spazio IP.

[3] Nella valutazione dell'opportunità o meno di installare un sistema di questo genere, bisogna considerare che i costi di manutenzione sono sicuramente preminenti su quelli di installazione. In quest'ottica, il TCO (costo totale di possesso, dall'inglese *Total Cost of Ownership*) di una soluzione basata su honeypot a bassa interazione è sicuramente più basso di altre soluzioni.

alliance, 2008]. Altri programmi degni di nota sono *Honeytrap* e *Amun*; il primo, ispirato a *Nepenthes* e presentato in [Werner, 2008], implementa alcune tecniche innovative. *Amun*, invece, è un prodotto completamente nuovo, scritto in Python, che rende molto semplice la scrittura di nuovi moduli di vulnerabilità; è reperibile presso [Göbel, 2008a].

3.2.2 Honeypot ad alta interazione

Una honeypot ad alta interazione (*high-interaction honeypot*) utilizza un sistema reale, non simulato, con attivi dei servizi vulnerabili. La honeypot può essere realizzata su una macchina fisica, eventualmente dotata di qualche meccanismo per il ripristino periodico[4], o come macchina virtuale. Al contrario della soluzione precedente, la fedeltà che si ottiene è in questo caso massima: l'attaccante interagisce con il sistema vulnerabile vero e proprio, non con una simulazione. Per contro, i costi di deployment e di manutenzione di una soluzione di questo genere sono sicuramente molto più elevati.

In una honeypot ad alta interazione l'attaccante, una volta sfruttata la vulnerabilità, ottiene a tutti gli effetti il controllo completo del sistema. Per evitare che le honeypot vengano usare come teste di ponte per eseguire altri attacchi o che partecipino in attacchi coordinati collettivi (ad esempio un DoS) è opportuno limitare il traffico di rete in uscita dalla honeynet. La soluzione comunemente adottata prevede l'utilizzo di un gateway tra la honeynet e Internet; un prodotto tipicamente utilizzato è *HoneyWall*, presentato in [The Honeynet project, 2005] e alla base della costruzione delle cosiddette honeynet di terza generazione di [Balas e Viecco, 2005]. *HoneyWall* è di fatto un transparent bridge con capacità di analisi e filtraggio dei pacchetti.

Come si è già detto, le honeypot possono essere implementate su macchine fisiche o sfruttando la virtualizzazione. In questo secondo caso, è possibile appoggiarsi a soluzione standard quali *VMWare* [VMWare, Inc., 2008], *Xen* [XenSource, 2008], *VirtualBox* [SUN Microsystems, 2008] o *qemu* [Bellard, 2005]. In alternativa, *Argos* abbina la virtualizzazione, basata su *qemu*, all'uso di tecniche di analisi per rilevare tempestivamente un attacco e registrare lo stato della macchina virtuale (mappa di memoria, ecc.); Argos è stato proposto inizialmente in [Portokalidis e altri, 2006].

3.2.3 Applicazioni specifiche

Di recente, sono state elaborate alcune tecniche per studiare e combattere fenomeni specifici, che non riguardano necessariamente la sicurezza in senso stretto. Queste tecniche sfruttano sempre più concetti ed idee implementati nelle honeypot, riadattati alla situazione specifica.

Una *spamtrap* è un indirizzo email pubblicato su una pagina web; l'indirizzo è camuffato (es. utilizzando i CSS) in modo che gli utenti che visitano normalmente il sito non possano vederlo, ma rimanga comunque presente nella pagina in formato

[4] È infatti necessario riportare periodicamente il sistema ad uno stato "pulito", poiché spesso diversi malware bloccano gli attacchi dei loro concorrenti e danneggiano in vario modo l'installazione. Questa operazione, nel caso di macchine fisiche, può essere eseguita dotando il sistema di un hardware dedicato (*restore card*), con dei meccanismi di disk imaging o sfruttando tecnologie quali PXE (*Preboot Execution Environment*) per il caricamento del sistema operativo dalla rete.

machine-readable. I programmi che periodicamente passano al setaccio la rete alla ricerca di indirizzi email (*harverster*) lo intercettano e, con ogni probabilità, iniziano a bersagliarlo di posta indesiderata. Poiché una spamtrap non riceve posta normale, tutti i messaggi ricevuti possono essere considerati spam, e utilizzati, ad esempio, per addestrare un filtro antispam o per studiare con un approccio statistico i messaggi.

Al contrario delle honeypot tradizionali, che aspettano passivamente di essere attaccate, le *client honeypot* vanno attivamente alla ricerca di server fraudolenti che attaccano i sistemi collegati. Una client honeypot interagisce, mascherata da normale client, con un server remoto, esaminando la sessione per determinare se e quando si verifica un attacco. Finora, la ricerca si è concentrata sui browser web, ed ha portato allo sviluppo di diversi toolkit e prodotti di analisi; citiamo *Capture-HPC* [The Honeynet project, 2008a], un prodotto ad alta interazione, e *HoneyC* [Seifert e altri, 2007b; The Honeynet project, 2008b], a bassa interazione. Un'analisi delle minacce presentate dai server web scoperte con questi strumenti è presentata in [Seifert e altri, 2007a].

3.3 Uso e deployment

Tipicamente, un'installazione di produzione di una honeynet prevede la combinazione di sistemi ad alta e bassa interazione; se l'obiettivo è lo studio di malware autoreplicante, si vuole raccogliere il maggior numero e varietà possibili di campioni. In tal caso è necessario implementare anche un sistema possibilmente automatizzato di raccolta ed analisi; per le honeypot ad alta interazione è inoltre necessario gestire il ripristino dei sistemi ad intervalli regolari. Degli esempi di deployment misti sono trattati in [Rajab e altri, 2006] e in [Zhuge e altri, 2007], oltre che nel Capitolo 4, in cui viene presentata l'infrastruttura costruita in questo lavoro.

Perché i dati raccolti siano statisticamente significativi è importante avere delle honeypot distribuite in modo uniforme sia geograficamente, sia in termini di spazio di indirizzamento IP. Per poter acquisire una buona comprensione dello stato delle minacce in rete sarebbe necessario avere a disposizione un gran numero di sensori. Nella pratica questo non è generalmente realizzabile, e si è quindi costretti a lavorare su dati necessariamente parziali. Si noti che la mancata copertura di un'area geografica o di una subnet può impedire il rilevamento di un attacco: l'acquisizione di una buona conoscenza a livello globale non può prescindere dall'analisi delle minacce diffuse localmente. Come mostrato da [Pouget e altri, 2005], è essenziale una buona copertura dei sensori per avere un quadro delle attività locali, che spesso sono particolarmente significative.

Per ovviare alle difficoltà di implementazione e gestione di un'infrastruttura centralizzata su larga scala sono state proposte diverse soluzioni, principalmente basate sul concetto di *honeypot distribuite*. L'approccio più comune è l'abbinamento ad una honeynet centralizzata (generalmente ad alta interazione) di una serie di *riflettori*, sistemi a basso costo che ne distribuiscono il traffico su larga scala in Internet. Questa tecnica, inizialmente presentata in *Collapsar* [Jiang e altri, 2006], permette di ridurre i costi di implementazione, rendendo al contempo più difficile il blocco di specifiche honeypot mediante blacklist e facilitando la correlazione dei dati raccolti. La stessa tecnica è utilizzata anche da *Honey@Home* [Antonatos e altri, 2007]; sviluppato nell'ambito del progetto NoAH [The NoAH project, 2008], questo

prodotto mira a costruire una vasta infrastruttura distribuita di sensori in modo cooperativo, sfruttando l'abbondanza di banda presente solitamente nelle connessioni residenziali a banda larga.

3.4 Problemi e criticità

Il deployment di una honeypot con successo è vincolato da alcuni problemi. È *essenziale* che la posizione delle honeypot sia mantenuta segreta: in caso contrario, un attaccante potrebbe cercare di disabilitarle (ad esempio lanciando un DoS), o di manipolare i dati raccolti eseguendo degli attacchi in modo mirato. In effetti, la sola presenza di una honeypot espone ad un rischio di ritorsioni; è quindi importante progettare con cura l'infrastruttura di rete correlata per evitare che eventuali attacchi alla honeynet possano influenzare segmenti di rete di produzione.

L'installazione di una honeypot espone ad un insieme di potenziali problemi legali. Come già citato, se si utilizzano honeypot ad alta interazione, per un certo periodo di tempo il malware è di fatto in esecuzione in modo non supervisionato sul sistema e ne ha potenzialmente il controllo completo. È indispensabile limitare il più possibile il rischio che dalla honeypot partano degli attacchi verso sistemi esterni; generalmente questo viene ottenuto frapponendo un gateway [The Honeynet project, 2005] tra la honeynet e Internet, che limita la quantità di pacchetti in uscita. Questo problema è particolarmente significativo per le iniziative di ricerca, che potrebbero essere ritenute responsabili di eventuali danneggiamenti a sistemi esterni in seguito ad esempio a un DoS. Questo problema, in particolare, concorre sicuramente ad aumentare il TCO di una soluzione ad alta interazione. Si noti però che limitare il traffico in uscita può rendere la honeypot rilevabile dall'esterno: come mostrato in [Zou e Cunningham, 2006], è possibile costruire un bot in grado di individuare automaticamente la presenza di una honeypot in base alle limitazioni sul traffico inviato. I lavori di [Dornseif e altri, 2004] e [Holz e Raynal, 2005] hanno mostrato come sia possibile rilevare via software una honeypot, specialmente se fa uso della virtualizzazione.

Capitolo 4

Apparato sperimentale

By believing passionately in something that still does not exist, we create it. The nonexistent is whatever we have not sufficiently desired.

Franz Kafka

Per poter studiare in modo oggettivo il comportamento e le caratteristiche delle botnet è necessario raccogliere dei dati. Il modo più immediato, che si è deciso di mettere in pratica, è l'acquisizione dei campioni di malware *in the wild* e lo studio del loro comportamento. In questo capitolo verrà descritto HIVE (*Honeypot Infrastructure in Virtualized Environment*), l'infrastruttura costruita per la raccolta, catalogazione ed analisi dei campioni. Il capitolo successivo, invece, si occuperà dell'interpretazione dei dati acquisiti. È stata posta particolare cura per garantire la completa riproducibilità dell'esperimento. A tale proposito, la costruzione di HIVE è ampiamente documentata in questo capitolo; i software utilizzati sono tutti Open Source, così come i programmi e gli script che sono stati realizzati durante questo lavoro. Tutto il codice realizzato è pubblicato nell'Appendice A.

4.1 Architettura

In Figura 4.1 è mostrata l'architettura del sistema sotto forma di schema a blocchi. HIVE è strutturato su tre livelli: honeypot, raccolta dei campioni ed analisi; l'aver

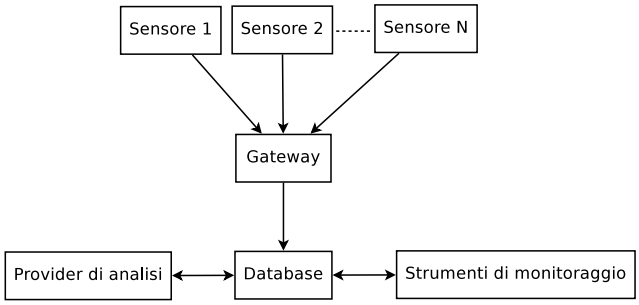


Figura 4.1: Architettura del sistema.

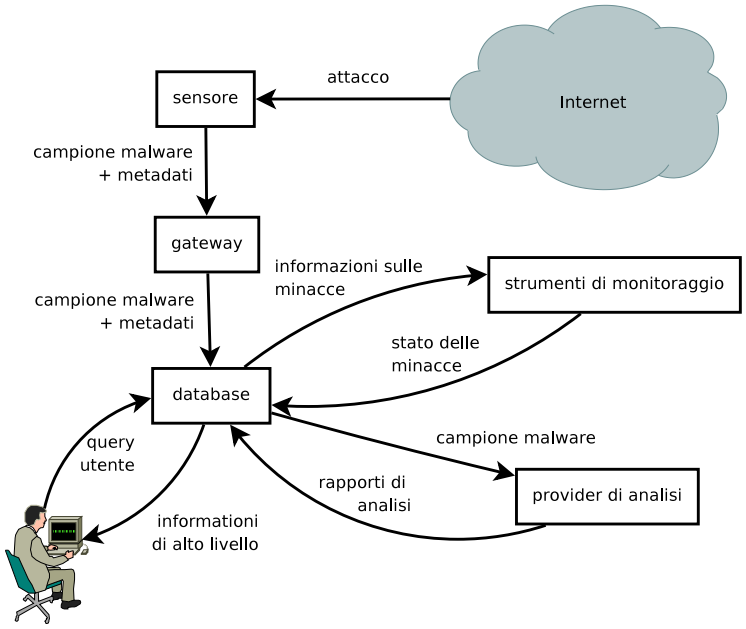


Figura 4.2: Ciclo vita del malware in HIVE.

disaccoppiato i diversi stadi semplifica l'implementazione e rende il sistema meglio scalabile. Inoltre, poiché le honeypot sono fisicamente separate dallo strato di analisi attraverso il gateway, non c'è rischio diretto di compromissione o danneggiamento dei dati raccolti. Il ciclo di vita tipico di un campione nel sistema è mostrato in Figura 4.2.

Uno schema della rete realizzata è mostrato in Figura 4.3 e sarà descritto in dettaglio nelle prossime sezioni. Sono state utilizzate tre macchine fisiche per costruire l'intera infrastruttura:

- *mercurio* gestisce l'acquisizione dei campioni (sensori);
- *venere* si occupa della catalogazione e dell'analisi (gateway, database);
- *terra* esegue un monitoraggio attivo delle botnet individuate.

4.2 Raccolta ed invio dei campioni

La raccolta dei campioni avviene attraverso delle honeypot, che occupano uno spazio di indirizzamento inutilizzato (*darknet*). Per raccogliere un maggior numero di campioni si è scelto di utilizzare sia honeypot ad alta interazione sia a bassa interazione. Questa scelta si è rivelata vincente: è stato possibile constatare che, sebbene le honeypot ad alta interazione raccolgano un numero di campioni notevolmente superiore, quelle a bassa interazione permettono di ottenere campioni che raggiungono i nostri sistemi usando dei vettori d'attacco altrimenti impossibili o molto difficili da sfruttare[1].

[1] Ci riferiamo, in particolare, ai malware che colpiscono specifici prodotti proprietari (Symantec, HP OpenView, ecc.).

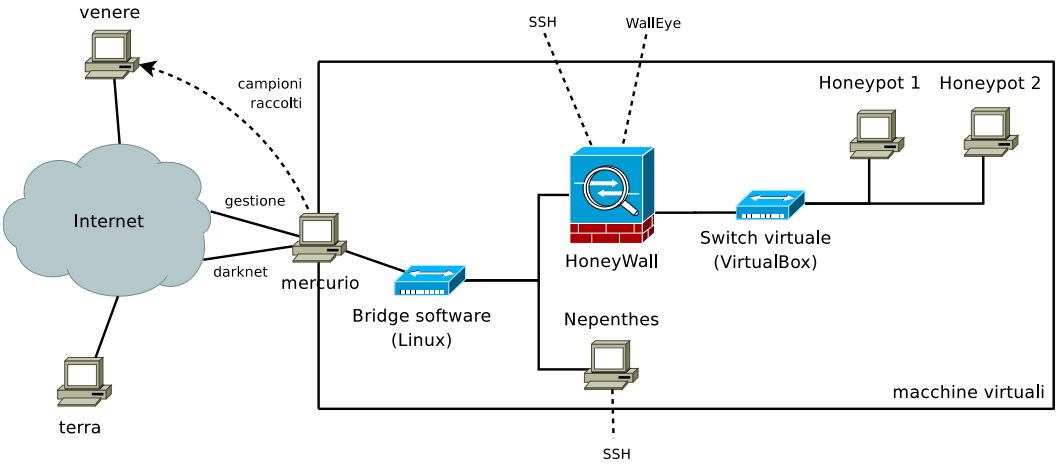


Figura 4.3: Schema della rete di raccolta dati realizzata.

L'intera infrastruttura di raccolta è stata realizzata attraverso macchine virtuali (VM), per massimizzare la semplicità di gestione e la flessibilità del sistema. Inoltre, implementando le honeypot ad alta interazione come macchine virtuali si eliminano quasi tutti gli svantaggi tipici di questa soluzione (già ampiamente descritti nella Sezione 3.2.2), facilitando il deployment dei sistemi e rendendo completamente automatizzabile l'estrazione dei campioni e la ricostruzione delle macchine. In generale, l'uso di macchine virtuali ha reso molto più semplice la gestione ed ha ridotto drasticamente i costi di implementazione, permettendo di consolidare i vari sistemi e costruire un'infrastruttura di rete complessa su una singola macchina fisica. Tutto ciò va a vantaggio anche della scalabilità: per aggiungere altre honeypot è sufficiente creare nuove VM (posto che la macchina host abbia sufficienti risorse).

Tra le honeypot ad alta interazione e la rete esterna è stato posto un filtering bridge (anch'esso implementato come macchina virtuale). Questo sistema, completamente trasparente[2], registra tutte le connessioni di rete in transito e le analizza con un IDS[3]. In questo modo si ottengono statistiche in tempo reale sull'utilizzo della honeynet e su eventuali attacchi subiti. Il bridge non interferisce con il traffico in ingresso: si vuole infatti che il malware arrivi intatto alle honeypot e le infetti con successo. Viene invece limitato il numero di pacchetti in uscita (es. a 10 pacchetti l'ora) dalla honeynet e sono bloccati gli attacchi noti; l'obiettivo è evitare la possibilità che le honeypot partecipino attivamente ad un attacco DoS, o che siano utilizzate come 'testa di ponte' per attaccare altri sistemi[4]. D'altra parte, non è possibile

[2]Un bridge opera al livello due della pila ISO/OSI; non possiede indirizzi IP ed è quindi completamente invisibile ai livelli superiori. In [Dornseif e altri, 2004] è presentata una tecnica per individuare un sistema di questo tipo; inviando PDU costruite in modo opportuno e controllando che non vengano alterate durante il percorso è infatti possibile ricostruire la presenza di un filtering bridge. Non è nota attualmente l'esistenza di malware *in the wild* che sfrutti questa tecnica.

[3]Un IDS (Intrusion Detection System) è, in questo contesto, un componente software che rileva gli attacchi e le minacce alla sicurezza presenti in un flusso di rete esaminando i pacchetti che lo compongono.

[4]Queste precauzioni si inseriscono nel più ampio contesto della *legal liability*, un tema significativo in questo settore di ricerca. Per evitare di esporsi a possibili conseguenze legali bisogna minimizzare il traffico maligno in uscita dalle nostre macchine; in caso contrario l'eventuale danneggiamento

bloccare *tutti* i pacchetti in uscita a prescindere; questo renderebbe inefficaci molti exploit, impedendo alla maggior parte dei malware di completare un'infezione con successo e vanificando quindi lo scopo stesso delle honeypot.

Prima di mettere la honeynet in produzione è stato necessario assicurarsi che i servizi esposti fossero effettivamente accessibili dall'esterno e vulnerabili. La darknet è stata quindi esaminata da una rete esterna, utilizzando diversi strumenti Open Source:

- il port scanner *Nmap* [Lyon, 2008] ha permesso di verificare i servizi di rete effettivamente esposti;
- con un vulnerability scanner *Nessus* [Tenable Network Security, 2008] sono state controllate le vulnerabilità presenti e sfruttabili;
- l'uso di *Metasploit* [Metasploit LLC, 2008], un framework automatizzato per l'exploit, ha infine consentito di portare a termine degli attacchi di prova.

4.2.1 Implementazione

Le soluzioni presenti in letteratura che fanno uso di honeypot virtualizzate, quali [The Artemis Team, 2008], generalmente si basano su *VMWare* [VMWare, Inc., 2008]. Per costruire HIVE si è scelto di utilizzare *VirtualBox* [SUN Microsystems, 2008], un prodotto Open Source multi-piattaforma relativamente recente con prestazioni simili a *VMWare*. La gestione delle macchine virtuali in *VirtualBox* può essere completamente automatizzata attraverso un'interfaccia a riga di comando o, nell'ultima versione 1.6.0, colloquiando con un web service. Questa caratteristica ha reso possibile la gestione in modo del tutto automatico delle macchine virtuali attraverso semplici script. Rispetto a *Xen* [XenSource, 2008], un'altro prodotto Open Source molto diffuso, *VirtualBox* può eseguire direttamente dei sistemi operativi non modificati (quali i sistemi Windows di Microsoft, di cui non è disponibile il sorgente) senza bisogno di un supporto hardware dedicato. È anche stato preso in considerazione l'utilizzo di libvirt [AA. VV., 2008], che fornisce un'interfaccia comune verso diversi motori di virtualizzazione; purtroppo al momento libvirt non supporta *VirtualBox*, anche se la sua integrazione dovrebbe essere possibile, sfruttando l'API C++ o il web service. È stata compilata ed installata la versione di sviluppo di *VirtualBox* dal repository SVN (revisione 7692).

Le macchine virtuali sono ospitate su *mercurio*, una macchina Linux con due interfacce di rete. L'interfaccia di gestione permette di accedere al sistema via SSH[5] e controllarlo da remoto in sicurezza, mentre il traffico verso la darknet[6] è veicolato da una seconda scheda di rete.

In totale, *mercurio* esegue quattro macchine virtuali:

---

di un sistema esterno potrebbe essere fonte di problemi. Questo problema è stato trattato più in dettaglio nella Sezione 3.4.

[5] SSH (Secure Shell) è un diffuso protocollo di rete per l'amministrazione remota di sistemi Unix; al contrario di telnet, SSH stabilisce una canale sicuro con il sistema remoto utilizzando degli schemi di crittografia forte.

[6] Nel nostro caso, la darknet era composta da tre indirizzi pubblici contigui di classe B; avevamo inoltre a disposizione due indirizzi per le interfacce di gestione di *mercurio* e *venere*. Gli indirizzi effettivi non saranno resi noti, per evitare eventuali ritorsioni o azioni di blacklist.

- una sistema Windows XP;
- una macchina Windows 2000 Server;
- un sistema Debian GNU/Linux 'etch' 4.0 che esegue *Nepenthes*;
- un sistema *HoneyWall* 'roo' 1.3

Le due macchine Windows sono le honeypot ad alta interazione; si tratta di installazioni standard del sistema operativo, con tutti i servizi di rete abilitati con le configurazioni di default e senza alcuna patch di sicurezza applicata. È stato deciso l'abbinamento di un sistema server a Windows XP (ad oggi il sistema client più diffuso), in modo da poter catturare anche il malware che attacca in modo specifico servizi disponibili sono su Windows 2000 Server, quali il web server IIS.

Le VM sono collegate ad uno switch virtuale (la *internal network* di *VirtualBox*); per raggiungere la rete esterna passano da *HoneyWall*, che funge da filtering bridge come descritto in precedenza. *HoneyWall* può essere amministrato via SSH o con WallEye, un'interfaccia web via HTTPS; questi servizi sono resi accessibili in modo sicuro sfruttando una funzione di *port forwarding* dalla macchina virtuale verso il sistema host[7]. Il collegamento tra *HoneyWall*, lo switch virtuale e la rete esterna è realizzato con un bridge software su *mercurio*, come mostrato in [The Linux Foundation, 2008].

Il progetto originale prevedeva anche l'utilizzo di *Sebek* sulle honeypot virtuali; si tratta di un *rootkit*[8], implementato come driver in spazio kernel, che monitora l'attività della macchina (tasti premuti, accesso al disco, utilizzo della rete, ecc.) e comunica con *HoneyWall*, Questo strumento si è rivelato poco stabile e fonte di diversi problemi ed incompatibilità (forse dovuti al sistema di virtualizzazione utilizzato) ed è quindi stato escluso dalle prove.

La soluzione adottata per la honeypot a bassa interazione è invece *Nepenthes* [Baecher e altri, 2006]. Questo programma permette di simulare molti servizi diversi, per molteplici sistemi operativi, ed è in grado di scaricare in automatico il malware che riceve. *Nepenthes* è eseguito come un normale programma su un sistema Linux, collegato anch'esso alla rete esterna attraverso un bridge software; anche in questo caso è utilizzato il *port forwarding* per permettere l'amministrazione via SSH del sistema. È stato testato anche *HoneyTrap* [Werner, 2008], un prodotto analogo, su piattaforma OpenBSD; si è però rivelato poco stabile, probabilmente a causa dello scarso supporto per OpenBSD da parte di *VirtualBox*.

La costruzione della rete virtuale e la configurazione delle honeypot è eseguita da uno script (`startnet.sh`, pubblicato nella Sezione A.1.1); questa soluzione ha

[7] Il *port forwarding* permette di dirottare il traffico di rete verso certe porte della macchina virtuale sull'interfaccia di loopback della macchina fisica. In questo modo, per amministrare *HoneyWall* via SSH (in ascolto sulla porta non standard 10022) basta collegarsi sulla 10022 di 127.0.0.1 (l'indirizzo di loopback, appunto).

[8] Un *rootkit* è un programma in spazio kernel (spesso implementato sotto forma di driver) che utilizza una serie di tecniche per nascondersi e non essere rilevabile. Generalmente un rootkit è uno strumento installato da un'attaccante non appena compromessa una macchina, per nascondere le sue tracce e cercare di preservare l'accesso al sistema senza destare sospetti. Nel caso in esame, Sebek utilizza le tecniche dei rootkit per non essere rilevato dal malware in esecuzione sul sistema o da un potenziale attaccante umano.

permesso di sperimentare diverse configurazioni in modo semplice e di assicurare la ripetibilità. La costruzione della rete avviene come segue:

1. si attiva il port forwarding per la gestione di *HoneyWall*, sulle porte 10022 (SSH) e 10443 (WallEye);
2. l'interfaccia esterna di *HoneyWall* è collegata ad una scheda di rete virtuale TUN/TAP, mentre l'interfaccia interna è collegata allo switch virtuale di *VirtualBox*;
3. per ogni honeypot virtuale, la sua scheda di rete è collegata allo switch virtuale;
4. si attiva il port forwarding per la gestione di *Nepenthes*, sulla porta 20022 (SSH);
5. l'interfaccia esterna di *Nepenthes* è collegata ad una seconda scheda di rete virtuale TUN/TAP;
6. viene creato un bridge software, utilizzando gli strumenti di bridging di Linux descritti in [The Linux Foundation, 2008], tra le due schede TUN/TAP e l'interfaccia fisica collegata alla darknet;
7. viene configurato il firewall del sistema host (basato su *iptables*) per bloccare tutti gli accessi (tranne ssh) dall'interfaccia fisica di gestione, e per permettere l'inoltro di pacchetti verso il bridge software.

Una volta configurata la rete le macchine virtuali vengono accese. L'estrazione e la raccolta dei campioni sono gestite da un secondo script (`resetvm.sh`, pubblicato nella Sezione A.1.2), che ad ogni esecuzione:

1. spegne la macchina virtuale;
2. confronta il contenuto del disco con un'immagine di riferimento (installazione "pulita");
3. registra in un file di log tutti i file aggiunti, modificati o eliminati;
4. copia eventuali file `.exe` aggiunti (probabili malware) in una directory di raccolta;
5. passa a `submit.py` gli eseguibili per l'invio via HTTP POST sull'interfaccia di gestione a *venere*;
6. clona l'immagine di riferimento in un nuovo disco e lo collega alla macchina al posto del precedente;
7. accende la macchina virtuale;

In origine la raccolta dei campioni era stata implementata con *MwWatcher*, uno strumento del progetto HoneyBow [The Artemis Team, 2008] che monitora gli accessi a disco e salva una copia degli eseguibili creati. Purtroppo questo programma si è rivelato poco affidabile, non riuscendo a rilevare la maggior parte delle modifiche. Si è scelto allora di confrontare "a freddo" il disco della macchina con un'immagine

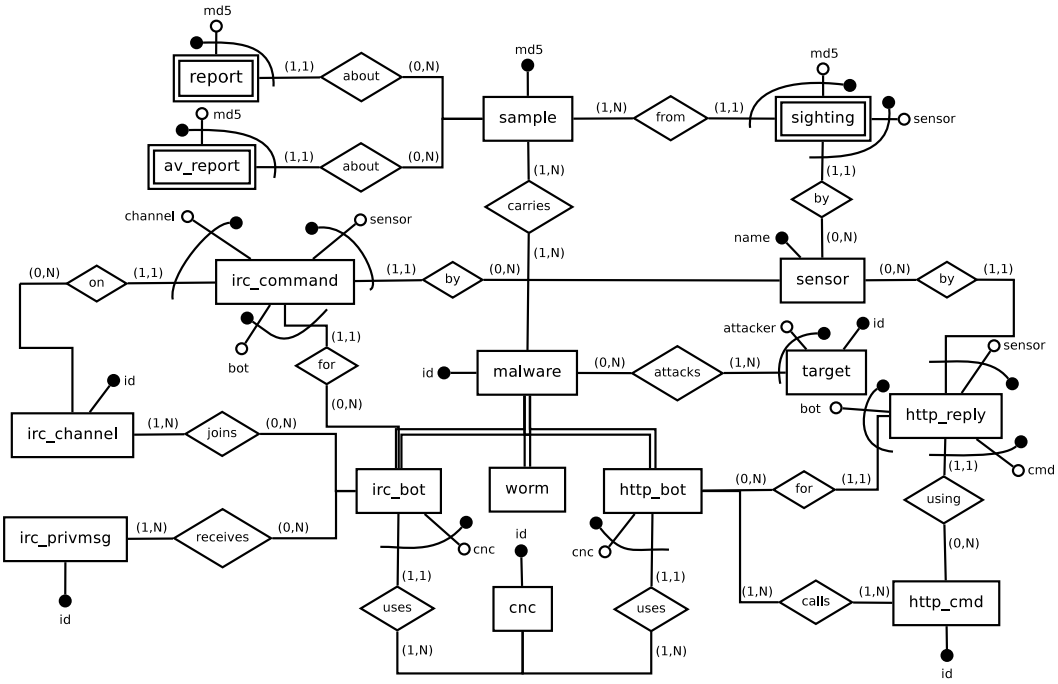


Figura 4.4: Diagramma ERA semplificato del database realizzato.

pulita, per avere la certezza di non perdere dei dati, usando *diff(1)*, un programma standard Unix.

L'invio dei campioni al gateway (che si trova su *venere*) è gestito da `submit.py`, uno script Python. Il campione viene spedito via HTTP POST insieme al alcuni metadati: hash MD5, nome del sensore, data di raccolta, nome, percorso, data e ora del file.

Per *Nepenthes* la procedura di raccolta è notevolmente semplificata, e sostanzialmente si riduce al prelevamento periodico dei campioni raccolti da una directory ed al loro invio con `submit.py`.

4.3 Ricezione, archiviazione e catalogazione

I campioni raccolti da *mercurio* sono inviati al gateway, che li inoltra al DBMS per la memorizzazione. Per semplicità, gateway e database sono stati implementati sulla stessa macchina, *venere*, anche se in generale ciò non è necessario (né consigliato). L'utilizzo di un database relazionale è stata una scelta obbligata per poter gestire in modo semplice ed efficace la grande mole di dati da memorizzare. Uno schema concettuale semplificato del database realizzato, secondo il modello ERA[9], è mostrato in Figura 4.4; la Tabella 4.1 raccoglie un dizionario delle entità presenti.

[9]Il modello ERA (Entità-Relazione-Attributo) è un modello per la rappresentazione concettuale dei dati ad un alto livello di astrazione. Viene spesso utilizzato nella prima fase della progettazione di una base di dati in cui è necessario tradurre le informazioni risultanti dall'analisi di un determinato dominio in uno schema concettuale.

nome	*descrizione*
av_report	rapporto della scansione antivirus di un campione
cnc	centro di controllo di una botnet
http_bot	tipo di malware che porta un bot HTTP
http_cmd	richiesta HTTP eseguita da un bot verso un cnc
http_reply	risposta HTTP dal cnc
irc_bot	tipo di malware che porta un bot IRC
irc_channel	canale IRC a cui un bot si collega
irc_command	comando IRC ricevuto dal cnc
irc_privmsg	messaggio privato ricevuto da un bot IRC
malware	istanza di codice infettivo
report	rapporto dell'analisi di un campione
sample	singolo campione di malware
sensor	punto di acquisizione del malware
sighting	avvirstamento di un campione su un sensore
target	connessione verso una potenziale vittima
worm	tipo di network malware generico

Tabella 4.1: Dizionario delle entità del database realizzato.

Lo schema logico ricavato dal modello concettuale, secondo il modello relazionale, è mostrato nella Tabella 4.2. Ogni campione è identificato dal suo hash MD5[10] all'interno dell'entità `sample`. I sensori disponibili sono invece identificati per nome in `sensor`. L'avvistamento di un campione su un sensore è registrato in `sighting`. Ogni campione, al suo inserimento, viene inviato ad alcuni servizi esterni (detti *provider*), che forniscono dei rapporti (`report`) sul suo comportamento. I campioni sono anche esaminati localmente con un antivirus; i rapporti degli antivirus, locali ed esterni, sono collezionati in `av_report`. In base al contenuto dei diversi rapporti il campione viene identificato e classificato come `worm`, `irc_bot` o `http_bot`; a seconda del tipo di campione viene deciso quali analisi aggiuntive eseguire e, se opportuno, sono registrate delle informazioni aggiuntive, quali il centro di controllo (`cnc`) e le connessioni stabilite verso potenziali vittime (`target`).

4.3.1 Analisi in sandbox

L'analisi dei campioni raccolti è stata affidata a due servizi online, entrambi gestiti da realtà accademiche. Questi sistemi analizzano il malware eseguendolo in una *sandbox* e studiandone il comportamento. Per mezzo di tecniche quali *API hooking* e *DLL injection* è possibile tracciare in modo preciso l'execution path del programma, registrando le chiamate di sistema effettuate ed il loro esito; in questo modo, ad esempio, si possono conoscere tutti gli accessi ai file del malware o i socket aperti.

[10] Pur essendo a conoscenza dei problemi di sicurezza di MD5, quali gli attacchi presentati in [Black e altri, 2006], si è scelto di usare comunque questo algoritmo di hash per identificare i campioni. L'uso di MD5 per identificare in modo univoco un file è ormai una pratica consolidata: si è preferito mantenere la compatibilità con altre soluzioni piuttosto che evitare la remota possibilità di una collisione forgiata.

relazione	*attributi*	*note*
av_reports	**id**, *md5*, scanner, scanner_version, signature_version, classification, processed	*md5* FK su SAMPLES(md5)
cncs	**id**, address, port, type, dnsname, country, active, latitude, longitude	
http_bots	**id**, *cnc*, last_seen	*cnc* FK su CNCS(id)
http_bots_ref	***http_bot***, ***http_cmd***	*http_bot* FK su HTTP_BOTS(id); *http_cmd* FK su HTTP_CMDS(id)
http_cmds	**id**, url, method, version, headers	
http_replies	**id**, timestamp, *http_bot*, *http_cmd*, *sensor*, status, reason, headers, response	*http_bot* FK su HTTP_BOTS(id); *http_cmd* FK su HTTP_CMDS(id); *sensor* FK su SENSORS(name)
irc_bots	**id**, *cnc*, hostname, realname, nickname, username, password, rfc_compliant, servername, last_seen	*cnc* FK su CNCS(id)
irc_bots_refc	***irc_bot***, ***irc_channel***	*irc_bot* FK su IRC_BOTS(id); *irc_channel* FK su IRC_CHANNELS(id)
irc_bots_refp	***irc_bot***, ***irc_privmsg***	*irc_bot* FK su IRC_BOTS(id); *irc_privmsg* FK su IRC_PRIVMSGS(id)
irc_channels	**id**, name, password, topics	
irc_commands	**id**, *irc_bot*, *irc_channel*, *sensor*, timestamp, command	*irc_bot* FK su IRC_BOTS(id); *irc_channel* FK su IRC_CHANNELS(id); *sensor* FK su SENSORS(name)
irc_privmsgs	**id**, type, message	
reports	**id**, *md5*, provider, available, task_id, analysis_id, password, processed, report, pcap	*md5* FK su SAMPLES(md5)
samples	**md5**, size, sample	
samples_ref	***sample***, type, **malware_id**	*sample* FK su SAMPLES(md5)
sensors	**name**, os, ip, low_interaction, notes	
sightings	**id**, *md5*, *sensor*, experiment, filename, filepath, filedate	*md5* FK su SAMPLES(md5); *sensor* FK su SENSORS(name)
targets	**id**, address, port, protocol, attacker, last_seen, dnsname, established	
worms	**id**, name	

Tabella 4.2: Schema logico del database secondo il modello relazionale.

Combinando questi risultati con un tracciato del traffico di rete effettuato si ottiene una fotografia piuttosto precisa del malware in esame. Questo tipo di analisi è detto *analisi comportamentale*, poiché studia il malware durante la sua esecuzione e lo classifica in base alle azioni che compie; questo sistema ci permette di ottenere un quadro dettagliato di *cosa* fa il campione "ai morsetti esterni", limitato però all'esecuzione presa in esame "qui e ora". Per contro, l'*analisi statica* analizza l'immagine eseguibile (es. con un antivirus, un decompilatore, ecc.) a freddo; questa tecnica può, in generale, vedere *tutti* gli execution path del malware, ma è estremamente laboriosa e può essere resa inefficace cifrando od offuscando il binario. Si noti che anche le sandbox di analisi possono essere rilevate: un malware potrebbe adottare delle contromisure per rendere più difficoltoso il suo studio, come mostrato in [Oberheide, 2008] e in [Ormandy, 2007]; non è nota, attualmente, l'esistenza di di malware "in the wild" che adotti tali tecniche.

I servizi utilizzati sono Anubis [International Secure Systems Lab, 2008], un'evoluzione di TTAnalyze presentata in [Bayer e altri, 2006], e CWSandbox [Sunbelt, 2008], introdotto in [Willems e altri, 2007]. Anubis è stato realizzato dall'International Secure Systems Lab con la collaborazione della Vienna University of Technology. Implementa la sandbox con una macchina virtuale *qemu* e fornisce un rapporto descrittivo in formato HTML, utile soprattutto per la consultazione manuale. CWSandbox, invece, è nato al Laboratory for Dependable Distributed Systems (University of Mannheim) ed è venduto commercialmente dalla Sunbelt Software. Per cautelarsi dalla possibilità di malware che attacchi direttamente una macchina virtuale (si veda ad esempio [Holz, 2007b]), la sandbox è implementata su macchine fisiche munite di *restore card* hardware che impediscono ogni modifica permanente. Oltre ad un rapporto HTML, CWSandbox mette anche a disposizione un documento XML contente tutti i dati dell'analisi e un tracciato del traffico di rete prodotto dal malware in formato PCAP; è quindi particolarmente comodo da interpretare in modo automatizzato per la classificazione dei campioni. I campioni inviati a CWSandbox sono anche analizzati in automatico con VirusTotal [Hispasec Sistemas, 2008], che fornisce i risultati della scansione con diversi motori antivirus. Altri servizi analoghi, che non sono stati utilizzati direttamente, ma che possono essere integrati nella nostra infrastruttura, sono Norman Sandbox [Norman, 2008] e Joebox [Joe Security, 2008].

In aggiunta all'analisi comportamentale, HIVE esamina i campioni ricevuti con una copia locale di ClamAV [Sourcefire, Inc., 2008], un motore antivirus Open Source. In questo modo è possibile classificare campioni poco interessanti (es. network worm) non appena vengono ricevuti, evitando così di sprecare ulteriori risorse per la loro analisi.

4.3.2 Implementazione

La ricezione dei campioni sul gateway avviene attraverso HTTP POST. Una pagina PHP (`upload.php`) si occupa di ricevere il campione, validarlo ed inviarlo al database per la registrazione; se esiste già un campione con lo stesso hash viene registrato solo il suo avvistamento in `sightings`.

Il DBMS utilizzato è *PostgreSQL* (alla versione 8.3), che supporta pienamente i vincoli di integrità referenziale e le proprietà ACID. È anche possibile definire stored

procedure e trigger da eseguire all'interno del database; questa funzione è stata sfruttata in modo esteso, in particolare per implementare la gestione dei rapporti. Lo schema fisico del database, comprensivo del DDL con la definizione delle diverse funzioni, è mostrato nell'Appendice B. Si è scelto di memorizzare i campioni come oggetti binari (BLOB) in un campo apposito della tabella `samples`.

All'inserimento di un campione in `samples`, il trigger `do_submit()` gestisce la scansione con ClamAV ed il suo invio (via HTTP POST) ad Anubis e CWSandbox, richiamando altre procedure. I campioni inviati sono processati da questi servizi in modo asincrono, a seconda del carico del sistema e del numero di richieste in coda; quando il rapporto è pronto viene inviata una email all'indirizzo specificato in fase di invio. Uno script (`imap2postgres.py`, pubblicato nella Sezione A.2.1) monitora periodicamente la casella di posta e registra in `reports` la disponibilità di nuovi rapporti. I rapporti ricevuti, infine, sono interpretati da `process_report.py`[11] (pubblicato nella Sezione A.2.2), che esegue, tra le altre, le seguenti operazioni:

- verifica che il campione non sia un worm noto (es. Allaple);
- legge tutte le connessioni di rete eseguite in uscita e i lookup DNS;
- a seconda del protocollo usato, registra il centro di comando in `cncs` e il malware nella tabella opportuna;
- legge e registra le informazioni specifiche del protocollo (es. per IRC il canale ed i messaggi privati) nelle tabelle opportune;
- marca il rapporto come "elaborato".

4.4 Monitoraggio

Le informazioni acquisite dai campioni forniscono tutti i dati necessari per il collegamento alla botnet. Utilizzando un programma opportuno, è possibile quindi infiltrare la botnet impersonando uno zombie; questo permette, tra l'altro, di ottenere informazioni sul modus operandi degli attaccanti, sulle dimensioni della rete e sugli attacchi perpetrati. A seconda del tipo di bot e del software utilizzato per gestire il centro di controllo l'infiltrazione può avere o meno successo, e le informazioni acquisibili sono ovviamente molto variabili.

Per il collegamento alle botnet IRC è stato usato *infiltrator*, un programma Python pubblicato in [Göbel, 2007] ed adattato per l'uso in HIVE. Il programma è stato integrato con il nostro database, implementando il caricamento automatico delle configurazioni dei bot e la registrazione delle loro attività. Poiché *infiltrator* è in grado di collegarsi a più botnet allo stesso tempo, è possibile monitorare un gran numero di reti con relativamente poco sforzo. Inoltre, il programma è in grado di rilevare alcuni comandi di aggiornamento delle botnet e prelevare il nuovo campione, che viene poi inserito nel database per la successiva analisi. Questa capacità permette

[11] In origine anche l'elaborazione dei rapporti era stata implementata come procedura all'interno del DBMS. Purtroppo questa soluzione si è rivelata poco affidabile: la nostra implementazione ha esposto diversi bachi nel motore procedurale Python di PostgreSQL, e siamo quindi stati costretti ad usare un programma esterno.

di seguire le botnet durante le migrazioni da un centro di controllo all'altro, che normalmente comportano un aggiornamento del malware.

Per studiare le botnet HTTP è stato scritto `httpmole.py`, un semplice programma che si collega periodicamente al centro di controllo e preleva la risposta HTTP. Analogamente ad *infiltrator*, è integrato con il database e può rilevare automaticamente alcuni aggiornamenti del malware. Il codice sorgente di `httpmole.py` è riportato nell'Appendice A.3.2.

Per evitare potenziali attacchi verso i sistemi che compongono HIVE, il monitoraggio è stato eseguito su una terza macchina, *terra*, collocata su una rete accademica.

Capitolo 5

Analisi dei dati

If your experiment needs statistics, you ought to have done a better experiment.

Ernest Rutherford

In questo capitolo verranno esaminati i dati raccolti nel corso dell'attività per cercare di caratterizzare meglio le botnet ed i diversi tipi di malware presenti in Internet. L'analisi si concentra su reti di tipo centralizzato: i diversi bot fanno a capo ad un unico centro di controllo (C&C, dall'inglese *Command and Control*), con cui scambiano dati usando i protocolli IRC o HTTP. Non sono stati invece esaminati bot *peer-to-peer* (P2P), trattati tra gli altri da [Grizzard e altri, 2007].

L'analisi vera e propria è stata eseguita operando delle query in SQL sulla base dei dati; si è utilizzato in modo esteso lo strumento delle viste[1], che permettono di filtrare i dati in modo granulare e dosare quindi il livello di dettaglio da prendere in considerazione.

Il grafico di Figura 5.2 mostra l'andamento dei campioni rilevati settimanalmente dai sensori, divisi per tipo. L'andamento è più o meno costante, e sembra mostrare un leggero decremento di Allaple nel tempo. I dati sono ovviamente parziali: per poter trarre delle conclusioni in qualche modo significative sarebbe necessario avere a disposizione un gran numero di dati provenienti da sensori sparsi uniformemente per il mondo.

5.1 Analisi antivirus

Grazie ai rapporti di VirusTotal ed ai risultati dell'analisi in locale con ClamAV si ottiene una fotografia dei diversi malware utilizzati per veicolare i bot. Come si può vedere anche dalla Figura 5.1, una parte consistente delle infezioni è ad opera di Allaple [F-Secure, 2006; Sophos, 2008], un network worm che attacca i sistemi Microsoft sfruttando dei buffer overflow nei servizi di rete e riproducendosi su condivisioni di rete non protette. Ai fini di questo studio, i worm sono principalmente "rumore di fondo", poiché non fanno parte di una botnet e si limitano a replicarsi ed eventualmente a danneggiare occasionalmente qualche sistema. La scrematura dei

[1] Una *vista* è una tabella virtuale, il cui contenuto è creato dinamicamente in base al risultato di una query prefissata. Il contenuto di una vista cambia in base ai dati referenziati dalla query.

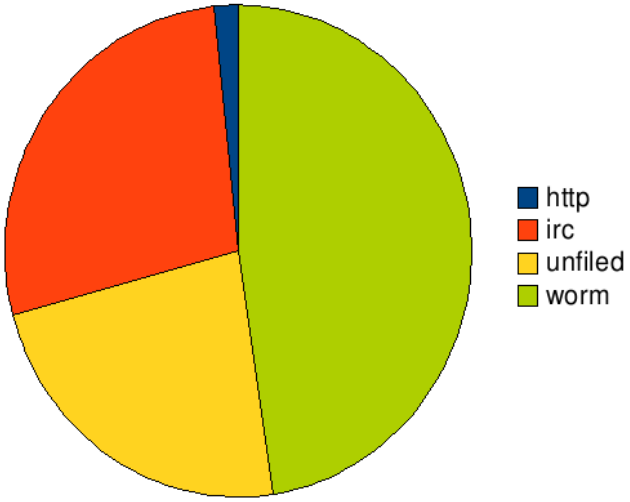


Figura 5.1: Distribuzione dei campioni raccolti per tipo.

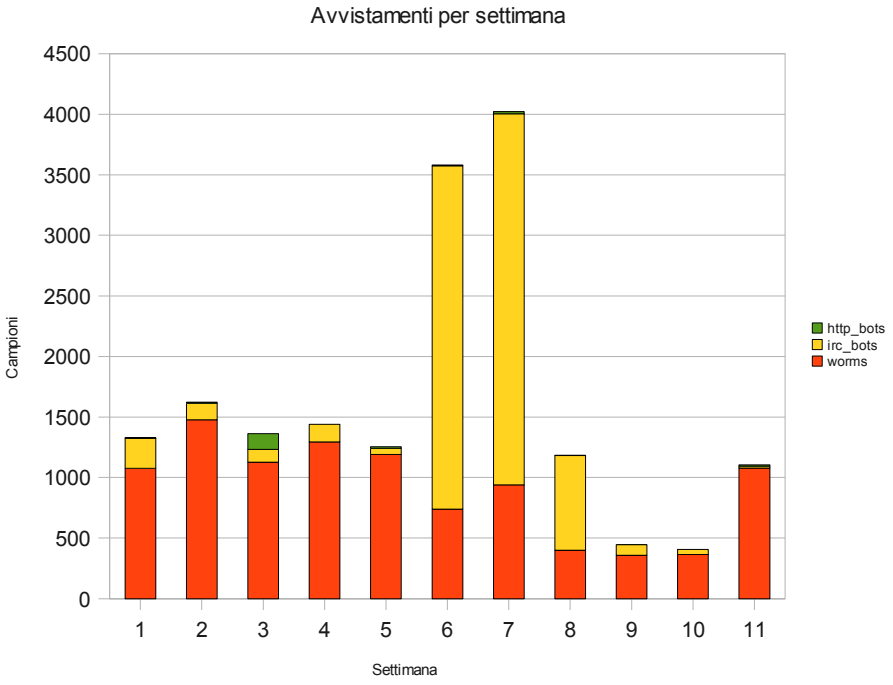


Figura 5.2: Grafico degli avvistamenti settimanali per tipo.

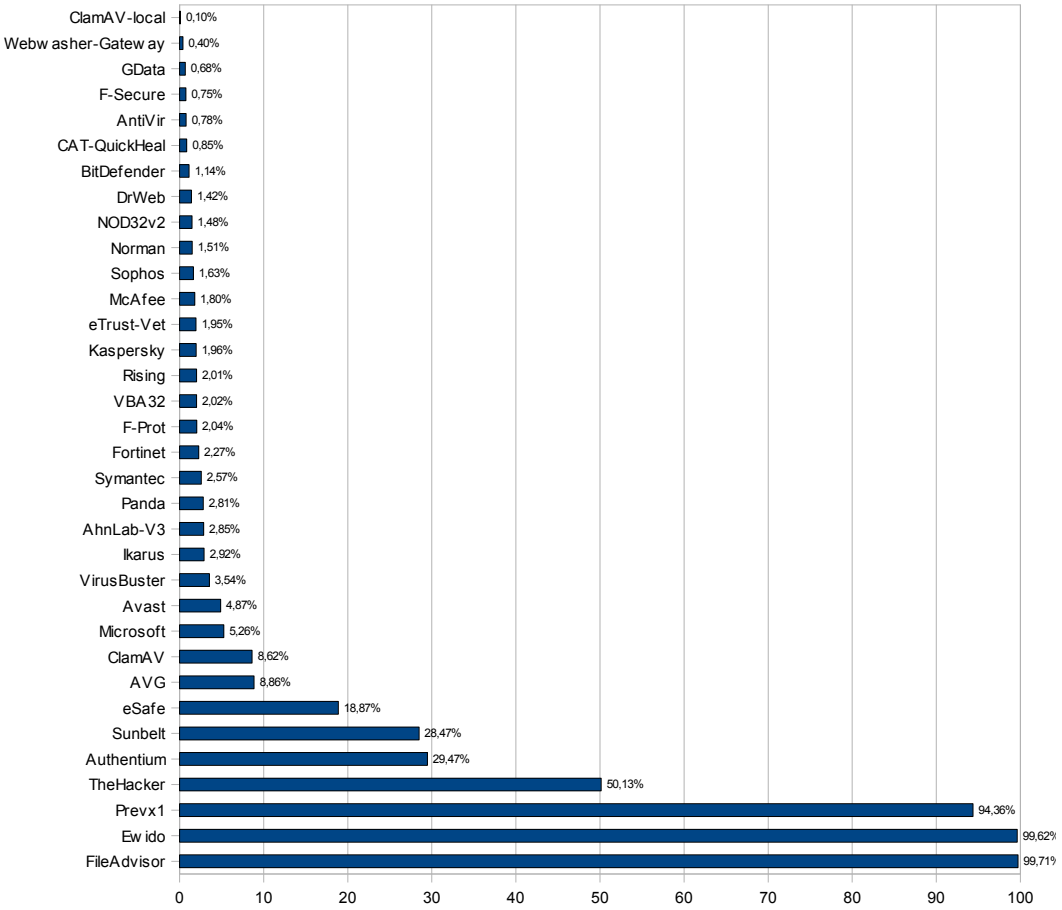


Figura 5.3: Percentuale di falsi negativi riportati dagli antivirus.

worm non è banale, poiché alcuni, come Allaple, sono polimorfici e non è possibile basarsi semplicemente sull'hash del campione, che cambia ad ogni mutazione. È possibile eseguire un'analisi comportamentale, che permetta di individuare la famiglia o il tipo di worm in base ad alcune caratteristiche comuni a tutte le sue mutazioni. Nel caso di Allaple, come mostrato in [Holz, 2007a], ad esempio, tutte le mutazioni del worm creano una mutex con lo stesso nome (`jhdheddfffffhjk5trh`) e si copiano sempre nel file `urdvxc.exe`. Questa tecnica, pur essendo molto accurata, produce un carico di lavoro estremamente elevato sui sistemi di analisi, che si trovano impegnati ad esaminare un gran numero di campioni sostanzialmente poco interessanti; per ovviare al problema, si è scelto di filtrare a priori i campioni identificati da ClamAV come varianti di Allaple ed escluderli dalle successive analisi. Questa soluzione, pur non essendo ideale (ClamAV produce un certo numero di classificazioni errate), permette di avere un buon compromesso tra risorse occupate e campioni rilevati, limitando il carico per i servizi di analisi.

Un interessante sottoprodotto di questo lavoro è una vasta mole di dati su campioni sicuramente maligni; in particolare, per la maggior parte dei malware abbiamo a disposizione le scansioni antivirus eseguite da diversi motori. Confrontando il numero di campioni etichettati come 'puliti' è possibile valutare in modo qualitativo

Figura 5.4: Mappa geografica dei centri di controllo rilevati.

l'accuratezza dei vari prodotti. In Figura 5.3 è appunto mostrata la distribuzione percentuale di falsi negativi per antivirus. Si noti che, poiché non abbiamo a disposizione rapporti di tutti gli antivirus su tutti i campioni, questi dati possono essere incompleti; inoltre, il dato relativo a *ClamAV-local* (il motore di scansione eseguito in locale sui nostri sistemi) non è direttamente confrontabile con gli altri, che provengono invece da VirusTotal.

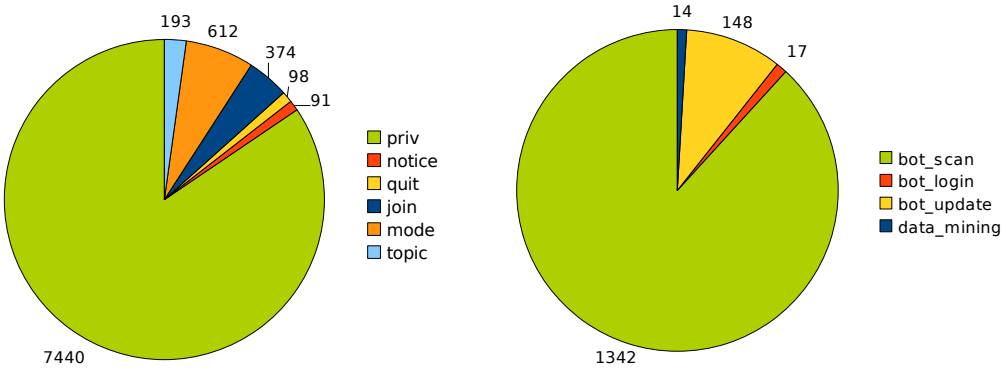
Per mostrare la facilità con cui è possibile rielaborare e manipolare i dati raccolti è stato realizzato un mashup con Google Maps [Google, 2008] che mostra la collocazione geografica dei diversi centri di controllo rilevati. Un esempio dell'output è presentato in Figura 5.4; come si può vedere, i C&C sono distribuiti in tutto il mondo.

5.2 Tipi di botnet

I diversi tipi di botnet conosciuti sono stati esaminati nella Sezione 2.3.1. In questa sezione verranno presentati i risultati sperimentali ottenuti studiando bot centralizzati che usano i protocolli IRC o HTTP per la comunicazione.

5.2.1 Bot IRC

Il centro di controllo di una botnet IRC esegue un server IRC a cui i diversi bot si collegano. In genere il *botherder* apporta diverse modifiche al server, per renderlo più robusto ad attacchi esterni e semplificarne la gestione; in quasi tutti i casi, viene disattivato il MOTD (il messaggio di benvenuto inviato ad ogni client alla connessione) e sono rimosse le notifiche delle JOIN e delle PART ai canali. Quest'ultima scelta è una misura per ridurre il traffico prodotto, che può essere sostanziale nel caso di botnet con migliaia di zombie collegati. Non tutti gli attaccanti, però, sono così



(a) Tipi di comandi IRC sul canale. **(b)** Ordini inviati agli zombie.

Figura 5.5: Analisi dei comandi inviati sul canale di controllo delle botnet IRC monitorate.

accorti: sono state individuate diverse botnet che utilizzavano server 'standard' che, alla connessione, annunciavano anche il numero di utenti connessi; un esempio è il seguente:

```
INFO :c 001 CcC-407378206 :Welcome to the b server CcC-407378206
INFO :c 002 CcC-407378206 :Your host is c, running version 5.5.2653
INFO :c 003 CcC-407378206 :This server was created Sep  9 2000 at 01:20:51 PDT
INFO :c 004 CcC-407378206 c 5.5.2653 aioxz abcdefhiklmnoprstuvxyz
INFO :c 251 CcC-407378206 :There are 1526 users and 1516 invisible on 1 servers
INFO :c 252 CcC-407378206 1 :operator(s) online
INFO :c 253 CcC-407378206 1 :unknown connection(s)
INFO :c 254 CcC-407378206 6 :channels formed
INFO :c 255 CcC-407378206 :I have 1526 clients and 0 servers
INFO :c 265 CcC-407378206 :Current local users: 1526 Max: 1526
INFO :c 266 CcC-407378206 :Current global users: 1526 Max: 1526
```

Diversi centri di controllo bloccano le connessioni provienienti da reti accademiche; il blocco (eseguito generalmente con un BAN verso l'indirizzo IP del sensore) è di frequente accompagnato da messaggi di insulti o di minaccia verso l'interlocutore. Poiché il nostro sistema di monitoraggio (*terra*) era collocato su una rete accademica, non è stato possibile infiltrare tali botnet.

Il controllo dei bot è generalmente eseguito attraverso l'invio di messaggi privati (PRIVMSG o NOTICE) agli zombie; alcune botnet, in alternativa, utilizzano il valore del TOPIC del canale di controllo come comando da eseguire. In Figura 5.5a è mostrata la distribuzione dei comandi IRC utilizzati dalle reti che abbiamo monitorato: come si può vedere, la stragrande maggioranza dei comandi utilizzati sono messaggi privati. La Figura 5.5b mostra invece le azioni ordinate agli zombie; gran parte dell'attività è dedicata all'espansione della botnet, seguita dall'esecuzione di eseguibili arbitrari (possibili aggiornamenti) o da azioni di data mining. È stato osservato, ad esempio, un botherder che ordinava agli zombie di andare alla ricerca di credenziali d'accesso a PayPal e eGold (servizi di pagamento online) o di prelevare le chiavi di registro contenenti i codici di registrazione di Diablo II (un popolare gioco online). Quest'ultima osservazione è in accordo con i risultati di altri ricercatori, quali [Göbel, 2008b], e mostra come i comportamenti dei botherder siano in continua evoluzione, sempre alla ricerca di nuove opportunità di profitto.

5.2.2 Bot HTTP

Un bot HTTP invia periodicamente una richiesta HTTP (tipicamente una GET) verso una URL del centro di controllo; in base al contenuto della risposta lo zombie esegue determinate azioni. Il controllo di questo tipo di bot è in genere implementato con soluzioni *ad hoc* (es. pagine PHP), che rendono difficile l'interpretazione della risposta in modo generale. Sono stati osservati i seguenti comportamenti:

- la risposta è un testo ASCII formattato in modo opportuno; un esempio catturato è il seguente (SENSOR_ADDRESS e CNC_ADDRESS sono rispettivamente gli indirizzi IP del sensore e del centro di controllo, in formato dotted-decimal):

```
IP: SENSOR_ADDRESS
ID: 16255236
CF: CNC_ADDRESS 92 bots hacked wbt.exe
SC: wkso 5 65
SC: d135 5 65
```

- la risposta è un URL (in ASCII, a volte offuscata in vari modi) che punta ad un eseguibile;
- la risposta è un eseguibile;
- la risposta è un blob binario inintellegibile.

In letteratura sono anche noti casi, che noi non abbiamo osservato, di bot che ricevono comandi codificati in base64; un esempio è il bot *BlackEnergy*, esaminato in dettaglio in [Nazario, 2007]. In alcuni casi, informazioni aggiuntive sono indicate attraverso opportune testate HTTP personalizzate.

Idealmente, si vorrebbe poter caratterizzare le risposte HTTP in modo automatico, estraendo i dati eventualmente codificati per interpretare il comportamento della botnet. In generale, questo è molto difficile da eseguire, a causa dell'estrema varietà rilevata nella forma e nei contenuti delle risposte; si ritene che, per poter eseguire un'analisi di qualche interesse, sarebbe quantomeno necessario implementare una forma di fingerprinting delle risposte, per poter distinguere ed aggregare i dati per categorie. In questo lavoro ci si è limitati ad una distinzione macroscopica tra le risposte contenenti testo in chiaro e quelle contenenti informazioni binarie; in questo secondo caso, se i dati sono identificati come un binario eseguibile[2] questo viene iniettato nel sistema di analisi come nuovo campione da studiare. Questa tecnica permette in molti casi di seguire la botnet quando il malware viene aggiornato per spostare il centro di controllo.

[2] Questa operazione è poco costosa, in quanto è possibile basarsi sul *magic number* del file, una stringa presente nei primi byte che identifica univocamente il tipo di file. Per gli eseguibili Windows in formato PE, ad esempio, i primi due byte contengono sempre la stringa 'MZ' (in ASCII), le iniziali di Mark Zbikowski, uno degli sviluppatori di MS-DOS e ideatore del formato.

Capitolo 6

Conclusioni

It may be that the gulfs will wash us down;
It may be that we shall touch the Happy Isles,
And see the great Achilles, whom we knew.
Though much is taken, much abides; and though
We are not now that strength which in old days
Moved earth and heaven, that which we are, we are —
One equal temper of heroic hearts,
Made weak by time and fate, but strong in will
To strive, to seek, to find, and not to yield.

Ulysses, l. 63–70
Alfred Tennyson

In questo lavoro è stata presentata un'infrastruttura per l'analisi ed il monitoraggio delle minacce in rete, con particolare attenzione verso le botnet ed il malware usato per costruirle. I dati raccolti mostrano la presenza di un fenomeno di enormi proporzioni che è stato possibile individuare e monitorare in modo relativamente semplice, nonostante il raggio fortemente limitato dei nostri sensori. Per poter acquisire una migliore comprensione delle minacce presenti e, soprattutto, aiutare la messa in opera di opportune contromisure, è auspicabile che strumenti ed infrastrutture analoghe ad HIVE siano sempre più diffuse, e che le diverse realtà interessate collaborino tra loro, condividendo dati ed esperienze per cercare di ottenere un quadro più completo. In quest'ottica, l'adozione di un modello dei dati unificato, come quello proposto in questo lavoro, potrebbe facilitare notevolmente la condivisione dei dati tra diversi sistemi.

L'infrastruttura realizzata può essere certamente migliorata ed ampliata sotto diversi aspetti. Al momento, i dati raccolti da *HoneyWall* sugli attacchi registrati non sono utilizzati in alcun modo; una loro integrazione con il database di HIVE permetterebbe di correlare ogni campione di malware con la vulnerabilità che ne ha permesso l'insediamento, e ricostruire quindi un quadro degli attacchi più utilizzati. L'analisi dei campioni, attualmente, dipende in gran parte dai risultati di CWSandbox, che viene utilizzato come provider principale; sarebbe auspicabile l'integrazione in modo automatico di altri motori di analisi, per avere un esame più accurato dei campioni e poter validare i risultati. La nostra infrastruttura non prevede alcun supporto per le botnet peer-to-peer, assumendo che tutti i campioni

raccolti facciano capo ad un centro di controllo stabile; poiché le botnet P2P sono sempre più diffuse un loro supporto, anche solo a livello di base, sarebbe utile per la classificazione dei campioni.

Per quanto riguarda l'implementazione effettiva delle honeypot, sarebbe interessante investigare la possibilità di un'interfaccia comune verso i diversi motori di virtualizzazione. È infine auspicabile la creazione di un'interfaccia di reportistica dei dati, che permetta di avere un quadro aggregato della situazione in modo semplice e veloce.; quest'ultima è, probabilmente, la limitazione principale di HIVE al momento.

La speranza è che questo lavoro possa contribuire ad una maggiore adozione delle honeypot nel mondo; come si è visto, le minacce presenti in rete sono reali ed in continua evoluzione: solo attraverso un costante monitoraggio ed una continua ricerca sarà possibile mantenere alta l'efficacia delle difese attuali e contribuire allo sviluppo di nuove soluzioni.

Appendice A

Codice sorgente

Questo capitolo raccoglie il codice dei programmi realizzati durante il lavoro. Tutti i programmi e gli realizzati sono reperibili all'indirizzo `http://netlab-mn.unipv.it/hive`.

A.1 Gestione delle macchine virtuali

Questa sezione raccoglie programmi e script in esecuzione su *mercurio* per gestire le macchine virtuali; sono stati descritti in dettaglio nella Sezione 4.2.1.

A.1.1 startnet.sh

```
1  #!/bin/bash
2
3  # VirtualBox user
4  VBOXUSER=davide
5  VBOXGRP=vboxusers
6
7  ##############
8  # HoneyWall #
9  ##############
10 HONEYWALL_VM=HoneyWall
11 # external interface
12 HONEYWALL_IF=tap2
13 # internal network
14 HONEYNET=honeynet
15 # forwarded ports
16 HONEYWALL_SSH=10022
17 HONEYWALL_WALLEYE=10443
18 # honeypots
19 HONEYPOT1_VM=Honeypot1
20 HONEYPOT2_VM=Honeypot2
21 HONEYPOT3_VM=Honeypot3
22
23 ##############
24 # Nephentes #
25 ##############
26 NEPENTHES_VM=Nepenthes
27 # external interface
28 NEPENTHES_IF=tap3
```

```
29 # forwarded ports
30 NEPENTHES_SSH=20022
31
32 ##############
33 # HoneyTrap #
34 ##############
35 HONEYTRAP_VM=HoneyTrap
36 HONEYTRAP_IF=tap4
37 HONEYTRAP_SSH=30022
38
39 # darknet-facing physical interface
40 DARKNET_IF=eth1
41
42 # darknet virtual bridge
43 DARKNET_BRIDGE=br1
44
45 # findprog <prog>
46 function findprog {
47         PROG=`which $1`
48         if [ ! -x $PROG ]; then
49                 PROG=`echo $1 | tr [A-Z] [a-z]`
50                 if [ ! -x $PROG ]; then
51                         echo "Cannot find $1, exiting."
52                         exit 1
53                 fi
54         fi
55         echo $PROG
56 }
57
58 # mktap <interface>
59 function mktap {
60         sudo tunctl -t $1 -u $VBOXUSER > /dev/null
61         sudo ip link set up dev $1
62 }
63
64 # rmtap <interface>
65 function rmtap {
66         sudo ip link set down dev $1
67         sudo tunctl -d $1 > /dev/null
68 }
69
70 # mkbridge <bridge> <interface...>
71 function mkbridge {
72         local BRIDGE=$1
73         sudo brctl addbr $BRIDGE
74         shift
75         for IF in $@; do
76                 sudo brctl addif $BRIDGE $IF
77         done
78         sudo ip link set up dev $BRIDGE
79 }
80
81 # rmbridge <bridge> <interface...>
82 function rmbridge {
83         local BRIDGE=$1
84         shift
85         sudo ip link set down dev $BRIDGE
86         for IF in $@; do
```

```
87                  sudo brctl delif $BRIDGE $IF
88          done
89          sudo brctl delbr $BRIDGE
90  }
91
92  # vmsetnic <vm> <nicnc> none
93  # vmsetnic <vm> <nicnr> <mac> nat
94  # vmsetnic <vm> <nicnr> <mac> hostif <dev>
95  # vmsetnic <vm> <nicnr> <mac> intnet <netname>
96  function vmsetnic {
97          local VMSTATUS=`$VBOXMANAGE -nologo showvminfo $1 |grep State:|
                awk '{ print $2 }'`
98          if [ $VMSTATUS == 'saved' ]; then
99                  echo "VM \"$1\" has a saved state, skipping update to
                        avoid corruption."
100         elif [ $VMSTATUS == 'running' ]; then
101                 echo "VM \"$1\" is currently running, cannot update."
102         else
103                 local ARGS
104                 local DEFARGS="-cableconnected$2 on -macaddress$2 $3"
105                 case "$4" in
106                         none)   ARGS="-nic$2 none";;
107                         nat)    ARGS="-nic$2 nat $DEFARGS";;
108                         hostif) ARGS="-nic$2 hostif -hostifdev$2 $5
                                $DEFARGS";;
109                         intnet) ARGS="-nic$2 intnet -intnet$2 $5
                                $DEFARGS";;
110                 esac
111                 $VBOXMANAGE -nologo modifyvm $1 $ARGS
112         fi
113 }
114
115 # startvm <vm> [headless]
116 function startvm {
117         if [ "$2" == 'headless' ]; then
118                 $VBOXHEADLESS -startvm $1 > /dev/null &
119         else
120                 $VBOXMANAGE -nologo startvm $1
121         fi
122 }
123
124 # stopvm <vm>
125 function stopvm {
126         $VBOXMANAGE -nologo controlvm $1 poweroff
127 }
128
129 function vmresetnatfwd {
130         # cleanup old forwarding settings
131         local OLDFWD=`$VBOXMANAGE -nologo getextradata $1 enumerate|
                grep pcnet|awk '{ print $2 }'|sed -e 's/,$//'`
132         for KEY in $OLDFWD; do
133                 $VBOXMANAGE -nologo setextradata $1 $KEY
134         done
135 }
136
137 # user manual p. 61
138 # vmnatfwd <vm> <nicnr> <servicename> <guestport> <hostport>
139 function vmnatfwd {
```

```
140         local NIC=$2
141         NIC=$[NIC-1]
142         $VBOXMANAGE -nologo setextradata $1 "VBoxInternal/Devices/pcnet
                /$NIC/LUN#0/Config/$3/Protocol" "TCP"
143         $VBOXMANAGE -nologo setextradata $1 "VBoxInternal/Devices/pcnet
                /$NIC/LUN#0/Config/$3/GuestPort" $4
144         $VBOXMANAGE -nologo setextradata $1 "VBoxInternal/Devices/pcnet
                /$NIC/LUN#0/Config/$3/HostPort" $5
145 }
146
147 function setup {
148         # allow virtualbox access to tap devices
149         if [ `ls -l /dev/net/tun | awk '{ print $4 }'` != $VBOXGRP ];
                then
150                 sudo chown :$VBOXGRP /dev/net/tun
151         fi
152
153         # honeywall managment
154         vmsetnic $HONEYWALL_VM 1 080027DF6E24 nat
155         vmresetnatfwd $HONEYWALL_VM
156         vmnatfwd $HONEYWALL_VM 1 roossh 22 $HONEYWALL_SSH
157         vmnatfwd $HONEYWALL_VM 1 walleye 443 $HONEYWALL_WALLEYE
158
159         # honeywall external
160         mktap $HONEYWALL_IF
161         vmsetnic $HONEYWALL_VM 2 080027659E52 hostif $HONEYWALL_IF
162
163         # honeywall internal
164         vmsetnic $HONEYWALL_VM 3 08002716711F intnet $HONEYNET
165
166         # honeypot1 internal
167         vmsetnic $HONEYPOT1_VM 1 0800272DCEEF intnet $HONEYNET
168
169         # honeypot2 internal
170         vmsetnic $HONEYPOT2_VM 1 080027D37916 intnet $HONEYNET
171
172         # honeypot3 internal
173         vmsetnic $HONEYPOT3_VM 1 080027DD0907 intnet $HONEYNET
174
175         # nephentes managment
176         vmsetnic $NEPENTHES_VM 1 080027F54BB0 nat
177         vmresetnatfwd $NEPENTHES_VM
178         vmnatfwd $NEPENTHES_VM 1 nepssh 20022 $NEPENTHES_SSH
179
180         # nephentes external
181         mktap $NEPENTHES_IF
182         vmsetnic $NEPENTHES_VM 2 0800275DE854 hostif $NEPENTHES_IF
183
184         # nephentes managment
185         vmsetnic $HONEYTRAP_VM 1 080027E584AD nat
186         vmresetnatfwd $HONEYTRAP_VM
187         vmnatfwd $HONEYTRAP_VM 1 honssh 22 $HONEYTRAP_SSH
188
189         # nephentes external
190         mktap $HONEYTRAP_IF
191         vmsetnic $HONEYTRAP_VM 2 080027EC060E hostif $HONEYTRAP_IF
192
193         # darknet bridge
```

```
194         mkbridge $DARKNET_BRIDGE $DARKNET_IF $HONEYWALL_IF
                $NEPENTHES_IF $HONEYTRAP_IF
195
196         # setup firewall
197         sudo iptables -P INPUT ACCEPT
198         sudo iptables -F INPUT
199         # filter everything but ssh on the management interface
200         sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j
                ACCEPT
201         sudo iptables -A INPUT -i eth0 -p tcp --dport ! 50022 -j DROP
202         sudo iptables -P OUTPUT ACCEPT
203         sudo iptables -F OUTPUT
204         sudo iptables -P FORWARD DROP
205         sudo iptables -F FORWARD
206         sudo iptables -A FORWARD -o $DARKNET_BRIDGE -j ACCEPT
207         sudo iptables -t nat -F
208 }
209
210 function cleanup {
211         # reset firewall
212         sudo iptables -P INPUT ACCEPT
213         sudo iptables -F INPUT
214         sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j
                ACCEPT
215         sudo iptables -A INPUT -i eth0 -p tcp --dport ! 50022 -j DROP
216         sudo iptables -P OUTPUT ACCEPT
217         sudo iptables -F OUTPUT
218         sudo iptables -P FORWARD ACCEPT
219         sudo iptables -F FORWARD
220         sudo iptables -t nat -F
221
222         rmbridge $DARKNET_BRIDGE $DARKNET_IF $HONEYWALL_IF
                $NEPENTHES_IF $HONEYTRAP_IF
223         rmtap $HONEYTRAP_IF
224         rmtap $NEPENTHES_IF
225         rmtap $HONEYWALL_IF
226 }
227
228 function start {
229         startvm $HONEYWALL_VM
230 #       startvm $NEPENTHES_VM
231         startvm $HONEYTRAP_VM
232 #       startvm $HONEYPOT1_VM
233         startvm $HONEYPOT2_VM
234         startvm $HONEYPOT3_VM
235 }
236
237 function stop {
238         stopvm $HONEYWALL_VM
239 #       stopvm $NEPENTHES_VM
240         stopvm $HONEYTRAP_VM
241 #       stopvm $HONEYPOT1_VM
242         stopvm $HONEYPOT2_VM
243         stopvm $HONEYPOT3_VM
244 }
245
246 VBOXMANAGE=`findprog VBoxManage`
247 VBOXHEADLESS=`findprog VBoxHeadless`
```

```
248
249 case "$1" in
250         start)
251                 echo -n "Starting VMs..."
252                 start
253                 echo "done."
254                 ;;
255         stop)
256                 echo -n "Stopping VMs..."
257                 stop
258                 echo "done."
259                 ;;
260         setup)
261                 sudo -v
262                 echo -n "Setting up networking..."
263                 setup
264                 echo "done."
265                 ;;
266         cleanup)
267                 sudo -v
268                 echo -n "Bringing down networking..."
269                 cleanup
270                 echo "done."
271                 ;;
272         *)
273                 echo "Usage: $0 {start|stop|setup|cleanup}"
274                 exit 1
275                 ;;
276 esac
277
278 exit 0
```

A.1.2 resetvm.sh

```
1  #!/bin/bash
2
3  VDIBASE=/home/davide/.VirtualBox/VDI
4  CURRENT_IMAGE="$1.vdi"
5  CLEAN_IMAGE="$1-clean.vdi"
6  VDITOOL=/home/davide/vditool
7  CLEAN_MOUNT=/tmp/clean.$$
8  CURRENT_MOUNT=/tmp/current.$$
9  TEMPDIFF=/tmp/diff.$$
10 TODAY="`date +%Y%m%d-%H%M`"
11 DIFFLOG="/home/davide/logs/$1-$TODAY.log"
12 STORE="/home/davide/store/$1/$TODAY"
13 SUBMIT="/usr/bin/python /home/davide/submit.py"
14 MOUNTVDI="/home/davide/mountvdi.sh"
15
16 VBOXMANAGE=`which vboxmanage`
17 if [ -z $VBOXMANAGE ]; then
18         VBOXMANAGE=`which VBoxManage`
19         if [ -z $VBOXMANAGE ]; then
20                 echo "Cannot find VBoxManage, exiting."
21                 exit 1
22         fi
23 fi
24
```

```
25 # mountvdi <vdi> <mountpoint>
26 function mountvdi {
27         local OFFSET=`$VDITOOL DUMP $1 | grep offData | sed 's/^.*=//'`
28         sudo mount -t ntfs -o loop,ro,noatime,noexec,offset=$[OFFSET
               +32256] $1 $2
29         rm -f *vditool*.log
30 }
31
32 # clonevdi <vdi> <vdi2>
33 function clonevdi {
34         $VBOXMANAGE -nologo clonevdi $1 $2
35 }
36
37 if [ ! -r "$VDIBASE/$CLEAN_IMAGE" ]; then
38         echo "ERROR: cannot find clean image '$VDIBASE/$CLEAN_IMAGE' 
               for VM '$1', exiting."
39         exit 1
40 fi
41
42 MY_CURRENT_IMAGE=`$VBOXMANAGE -nologo showvminfo $1|grep 'Primary
       master' | awk '{ print $3 }'`
43 if [ "$VDIBASE/$CURRENT_IMAGE" != "$MY_CURRENT_IMAGE" ]; then
44         echo "ERROR: current image for VM '$1' is '$MY_CURRENT_IMAGE', 
               but I was expecting '$VDIBASE/$CURRENT_IMAGE', exiting."
45         exit 1
46 fi
47
48 $VBOXMANAGE -nologo controlvm $1 poweroff
49 sleep 5
50 $VBOXMANAGE -nologo modifyvm $1 -hda none
51 $VBOXMANAGE -nologo unregisterimage disk $CURRENT_IMAGE
52
53 for i in `sudo /sbin/losetup -a|cut -f1 -d:`; do
54         sudo /sbin/losetup -d $i
55 done
56
57 mkdir -p $CLEAN_MOUNT $CURRENT_MOUNT
58 mountvdi "$VDIBASE/$CURRENT_IMAGE" "$CURRENT_MOUNT"
59 mountvdi "$VDIBASE/$CLEAN_IMAGE" "$CLEAN_MOUNT"
60
61 echo "Processing diff log for $1 at `date`" > $DIFFLOG
62 LANG=C diff -rq $CLEAN_MOUNT $CURRENT_MOUNT > $TEMPDIFF
63 echo >> $DIFFLOG
64 echo "Added:" >> $DIFFLOG
65 grep "^Only in $CURRENT_MOUNT" $TEMPDIFF |sed 's/^Only\ in\ /\"/'|sed '
       s/\:\ /\//'|sed 's/$/\"/' | xargs ls -ld --full-time >> $DIFFLOG
66 echo >> $DIFFLOG
67 echo "Removed:" >> $DIFFLOG
68 grep "^Only in $CLEAN_MOUNT" $TEMPDIFF |sed 's/^Only\ in\ /\"/'|sed 's
       /\:\ /\//'|sed 's/$/\"/' | xargs ls -ld --full-time >> $DIFFLOG
69 echo >> $DIFFLOG
70 echo "Changed:" >> $DIFFLOG
71 grep '^Files.*differ$' $TEMPDIFF | sed 's/Files.*\ and\ \//\"\//'|sed
       's/\ differ/\"/' | xargs ls -ld --full-time >> $DIFFLOG
72 echo >> $DIFFLOG
73 echo "Processing ended at `date`" >> $DIFFLOG
74
75 EXPERIMENT=$TODAY
```

```
76  SENSOR=$1
77
78  # warning: ugly hack follows...
79  mkdir -p $STORE
80  IFS='|'
81  LIST=`grep "^Only in $CURRENT_MOUNT" $TEMPDIFF |sed 's/^Only\ in\ //'|
        sed 's/\:\ /\//'|sed 's/$/|/'|grep exe`
82  for i in $LIST; do
83          cp -a `echo $i | tr -d '\n'` $STORE
84          FILE=`echo $i | tr -d '\n'`
85          unset IFS
86          FILENAME=`basename "$FILE"`
87          FILESIZE=`ls -l "$FILE" | awk '{ print $5 }'`
88          if [ "$FILESIZE" == '0' ]; then
89                  continue
90          fi
91          FILEDATE=`ls --full-time "$FILE" | awk '{ print $6 " " $7 " "
                $8 }'`
92          # hack: works only for /tmp/current.$$
93          FILEPATH=`dirname "$FILE"| sed 's@/tmp/current\.[0-9]*@@'`
94          $SUBMIT -e "$EXPERIMENT" -s "$SENSOR" -n "$FILENAME" -p "
                $FILEPATH" -d "$FILEDATE" -z "$FILESIZE" "$FILE"
95          IFS='|'
96  done
97  LIST=`grep '^Files.*differ$' $TEMPDIFF | sed 's/Files.*\ and\ \//\//'|
        sed 's/\ differ//'|sed 's/$/|/'|grep exe`
98  for i in $LIST; do
99          cp -a `echo $i | tr -d '\n'` $STORE
100         FILE=`echo $i | tr -d '\n'`
101         unset IFS
102         FILENAME=`basename "$FILE"`
103         FILESIZE=`ls -l "$FILE" | awk '{ print $5 }'`
104         if [ "$FILESIZE" == '0' ]; then
105                 continue;
106         fi
107         FILEDATE=`ls --full-time "$FILE" | awk '{ print $6 " " $7 " "
                $8 }'`
108         FILEPATH=`dirname "$FILE"| sed 's@/tmp/current\.[0-9]*@@'`
109         $SUBMIT -e "$EXPERIMENT" -s "$SENSOR" -n "$FILENAME" -p "
                $FILEPATH" -d "$FILEDATE" -z "$FILESIZE" "$FILE"
110         IFS='|'
111 done
112 unset IFS
113
114 rm -f $TEMPDIFF
115
116 sudo umount $CURRENT_MOUNT
117 sudo umount $CLEAN_MOUNT
118 rmdir $CURRENT_MOUNT $CLEAN_MOUNT
119
120 # switch back to the clean image
121 mv $VDIBASE/$CURRENT_IMAGE $VDIBASE/`basename $CURRENT_IMAGE .vdi`.old.
        vdi
122 clonevdi $CLEAN_IMAGE $CURRENT_IMAGE
123 $VBOXMANAGE -nologo registerimage disk $CURRENT_IMAGE
124 $VBOXMANAGE -nologo modifyvm $1 -hda $CURRENT_IMAGE
125 $VBOXMANAGE -nologo startvm $1
```

A.2 Gestione database

Questa sezione raccoglie programmi e script in esecuzione su *venere* che interagiscono con il DBMS.

A.2.1 imap2postgres.py

Il programma che segue esamina via IMAP i messaggi presenti in una casella email e preleva eventuali rapporti di Anubis e di CWSandbox, registrandoli poi nel database.

```
1  #!/usr/bin/python
2  from imaplib import IMAP4_SSL
3  from email.Parser import Parser
4  import re
5  import pg
6  import sys
7
8  file_r = re.compile('^Submitted file: ([0-9a-z]+)\r\n',re.M)
9  analysis_r = re.compile('^Analysis ID: ([0-9]+)\r\n',re.M)
10 password_r = re.compile('.*password=3D([a-z]+).\r\n',re.M)
11 result_r = re.compile('^Analysis result: (.*)\r\n',re.M)
12
13 error_r = re.compile('^Unfortunately there was an error while trying to
       analyze your file.\r\n', re.M)
14 taskid_r = re.compile('^You can find your report at http://analysis\.
       seclab\.tuwien\.ac\.at/result\.php\?taskid=([0-9a-z]+)\r\n',re.M)
15
16 def parse_report(report):
17         m = file_r.search(report)
18         if m == None:
19                 submitted_file = ''
20         else:
21                 submitted_file = m.group(1)
22
23         m = analysis_r.search(report)
24         if m == None:
25                 analysis_id = ''
26         else:
27                 analysis_id = m.group(1)
28
29         m = result_r.search(report)
30         if m == None:
31                 analysis_result = ''
32         else:
33                 analysis_result = m.group(1)
34
35         m = password_r.search(report)
36         if m == None:
37                 password = ''
38         else:
39                 password = m.group(1)
40
41         return submitted_file, analysis_id, analysis_result, password
42
43 def is_duplicate(db, md5):
44         q = "SELECT md5 FROM reports WHERE md5 = '%s' and provider = '
               CWSandbox'" %(md5)
45         if db.query(q).ntuples() > 0:
```

```
46                  return True
47          else:
48                  return False
49
50  def insert_report(db, md5, id, password):
51          q = "INSERT INTO reports (md5, provider, analysis_id, password,
                available) VALUES ('%s', 'CWSandbox', %s, '%s', 'true')"
                %(md5, id, password)
52          db.query(q)
53          return
54
55  def set_invalid(db, md5):
56          q = "UPDATE samples SET type = 'invalid' WHERE md5 = '%s'" %(
                md5)
57          db.query(q)
58          return
59
60  def parse_anubis(report):
61          m = error_r.search(report)
62          if m != None:
63                  return ''
64          m = taskid_r.search(report)
65          if m == None:
66                  return ''
67          return m.group(1)
68
69  def is_anubis(db, taskid):
70          q = "SELECT md5 FROM reports WHERE task_id = '%s' and provider
                = 'Anubis' and available = 'false'" %(taskid)
71          if db.query(q).ntuples() > 0:
72                  return True
73          else:
74                  return False
75
76  def update_anubis(db, taskid):
77          q = "UPDATE reports SET available = 'true' WHERE task_id = '%s'
                " %(taskid)
78          db.query(q)
79          return
80
81  parser = Parser()
82  conn = pg.connect(host='localhost', port=5432, dbname='malware', user='
        malware_user', passwd='malware_password')
83  server = IMAP4_SSL("mailserver.example.com", 993)
84  server.login('mailbox@example.com','email_password')
85  server.select('INBOX')
86  typ, data = server.search(None, 'ALL')
87  done_msg = []
88  invalid_msg = []
89  for num in data[0].split():
90          print "Processing message %s" %(num)
91          typ, data = server.fetch(num, '(RFC822)')
92          if data == None:
93                  print "cannot fetch message, skipping..."
94                  continue
95          if data[0] == None:
96                  print "cannot fetch message, skipping..."
97                  continue
```

```
98          mesg = parser.parsestr(data[0][1])
99          if mesg['sender'] == 'sandbox@informatik.uni-mannheim.de':
100                 body = mesg.get_payload()
101                 md5, id, result, password = parse_report(body)
102                 if(is_duplicate(conn, md5)):
103                         print "Duplicate report %s, skipping..." %(md5)
104                         done_msg.append(num)
105                         continue
106                 if(result != 'Processed'):
107                         print "Invalid report for %s: '%s', skipping...
                                " %(md5, result)
108                         invalid_msg.append(num)
109                         set_invalid(conn, md5)
110                         continue
111                 insert_report(conn, md5, id, password)
112                 done_msg.append(num)
113         elif mesg['From'] == 'analysis-noreply@seclab.tuwien.ac.at':
114                 body = mesg.get_payload()
115                 taskid = parse_anubis(body)
116                 if taskid == '':
117                         print "Invalid report, skipping..."
118                         invalid_msg.append(num)
119                         continue
120                 if is_anubis(conn, taskid):
121                         update_anubis(conn, taskid)
122                 else:
123                         print "Duplicate or unknown report, skipping...
                                "
124                 done_msg.append(num)
125         else:
126                 print "sender unknown, skipping..."
127 sys.stdout.write('Marking processed messages')
128 for num in done_msg:
129         server.copy(num, 'done')
130         sys.stdout.write('.')
131         sys.stdout.flush()
132 print
133 sys.stdout.write('Marking invalid messages')
134 for num in invalid_msg:
135         server.copy(num, 'invalid')
136         sys.stdout.write('.')
137         sys.stdout.flush()
138 print
139 sys.stdout.write('Cleaning up INBOX')
140 for num in done_msg:
141         server.store(num, '+FLAGS.SILENT', '\\Deleted')
142         sys.stdout.write('.')
143         sys.stdout.flush()
144 for num in invalid_msg:
145         server.store(num, '+FLAGS.SILENT', '\\Deleted')
146         sys.stdout.write('.')
147         sys.stdout.flush()
148 print
149 server.expunge()
150 server.close()
151 server.logout()
152 conn.close()
```

A.2.2 process_report.py

Il programma che segue interpreta il contenuto di un rapporto di CWSandbox ed aggiorna opportunamente il database con i nuovi dati acquisiti.

```
1  #!/usr/bin/python
2  import psycopg2
3  import xml.dom.minidom
4  from xml.dom.minidom import Node
5  import urllib2
6  import sys
7  import re
8
9  ALLAPLE_MUTEX_NAME = 'jhdheddfffffhjk5trh'
10 ALLAPLE_FILE_NAME = 'urdvxc.exe'
11 ALLAPLE_ID = 1
12
13 def getAttribute(attributes, key):
14     if attributes.has_key(key):
15         return attributes[key].value
16     else:
17         return None
18
19 def getText(node):
20     val = ''
21     for node3 in node.childNodes:
22         if node3.nodeType == Node.TEXT_NODE:
23             val += node3.data
24     return val
25
26 def process_virustotal(cur, report):
27         # SQL prepared queries
28     set_processed = 'UPDATE reports SET processed = true WHERE md5 = %s
           and provider::text = %s'
29     add_av_report = 'INSERT INTO av_reports (md5, scanner,
           scanner_version, signature_version, classification) VALUES (%s,
           %s, %s, %s, %s)'
30     process_av_report = 'SELECT process_av_report(%s)'
31
32     allaple_count = 0
33     root_n = xml.dom.minidom.parseString(report)
34     analysis_n = root_n.getElementsByTagName('analysis')[0]
35     md5 = getAttribute(analysis_n.attributes, 'md5')
36     virusscan_n = analysis_n.getElementsByTagName('virusscan_section')
           [0]
37     for scanner_n in virusscan_n.getElementsByTagName('scanner'):
38         scanner = getAttribute(scanner_n.attributes, 'name')
39         scanner_version = getAttribute(scanner_n.attributes, '
               application_version')
40         signature_version = getAttribute(scanner_n.attributes, '
               signature_file_version')
41         classification = getText(scanner_n.getElementsByTagName('
               classification')[0])
42         cur.execute(add_av_report, [ md5, scanner, scanner_version,
               signature_version, classification ])
43     cur.execute(set_processed, [ md5, 'VirusTotal' ])
44     cur.execute(process_av_report, [ md5 ])
45     return True
46
```

```
47 def process_cwsandbox(cur, report):
48         # SQL prepared queries
49     add_cnc = 'SELECT add_cnc(%s, %s, %s, %s)'
50     add_http_bot = 'SELECT add_http_bot(%s, %s)'
51     add_http_cmd = 'SELECT add_http_cmd(%s, %s, %s, %s, %s)'
52     add_irc_bot = 'SELECT add_irc_bot(%s, %s, %s, %s, %s, %s, %s, %s, %
           s)'
53     add_irc_channel = 'SELECT add_irc_channel(%s, %s, %s, %s)'
54     add_irc_privmsg = 'SELECT add_irc_privmsg(%s, %s, %s)'
55     add_target = 'SELECT add_target(%s, %s, %s, %s, %s, %s, %s)'
56     set_sample = 'SELECT set_sample(%s, %s, %s)'
57     set_processed = 'UPDATE reports SET processed = true WHERE md5 = %s
           and provider::text = %s'
58
59     dnsqueries = []
60     revdnsqueries = []
61     allaple_mutex = False
62     allaple_file = False
63     winfile_r = re.compile('^.*\\\\(.*)$')
64     root_n = xml.dom.minidom.parseString(report)
65     analysis_n = root_n.getElementsByTagName("analysis")[0]
66     md5 = getAttribute(analysis_n.attributes, 'md5')
67     analysis_date = getAttribute(analysis_n.attributes, 'time')
68     date_r = re.compile('^([0-9]+)/([0-9]+)/([0-9]+)\ (.*)$')
69     date_m = date_r.search(analysis_date)
70     if date_m != None:
71         month = date_m.group(1)
72         day = date_m.group(2)
73         year = date_m.group(3)
74         time = date_m.group(4)
75         analysis_date = "%s.%s.%s %s" %(day, month, year, time)
76         print "braindead date format, workaround activated; fixed date
               is '%s'" %(analysis_date)
77     processes_n = analysis_n.getElementsByTagName("processes")[0]
78     targets = []
79     malwares = []
80     for process_n in processes_n.getElementsByTagName("process"):
81         ## FILESYSTEM ##
82         filesys_n = process_n.getElementsByTagName("filesystem_section"
               )
83         if len(filesys_n) > 0:
84             copy_n = filesys_n[0].getElementsByTagName("copy_file")
85             for copy_op in copy_n:
86                 dest = copy_op.attributes['dstfile'].value
87                 m = winfile_r.search(dest)
88                 if m != None:
89                     name = m.group(1)
90                     if name == ALLAPLE_FILE_NAME:
91                         allaple_file = True
92         ## MUTEX ##
93         mutex_n = process_n.getElementsByTagName("mutex_section")
94         if len(mutex_n) > 0:
95             create_mutex_n = mutex_n[0].getElementsByTagName("
                   create_mutex")
96             for mutex in create_mutex_n:
97                 name = getAttribute(mutex.attributes, 'name')
98                 if name == ALLAPLE_MUTEX_NAME:
99                     allaple_mutex = True
```

```
100             if allaple_file and allaple_mutex:
101                 cur.execute(set_sample, [ md5, 'worm', ALLAPLE_ID ])
102                 cur.execute(set_processed, [ md5, 'CWSandbox' ])
103                 return True
104             ## WINSOCK ##
105             winsock_n = process_n.getElementsByTagName("winsock_section")
106             if len(winsock_n) > 0:
107                 conns_unk_n = winsock_n[0].getElementsByTagName("
                        connections_unknown")
108                 if len(conns_unk_n) > 0:
109                     conn_unk_n = conns_unk_n[0]
110                     for conn_n in conn_unk_n.getElementsByTagName("
                            connection"):
111                         for gethostbyname_n in conn_n.getElementsByTagName(
                                "gethostbyname"):
112                             gethostbyname_a = gethostbyname_n.attributes
113                             host = getAttribute(gethostbyname_a, '
                                    requested_host')
114                             address = [ getAttribute(gethostbyname_a, '
                                    resulting_addr') ]
115                             more_n = gethostbyname_n.getElementsByTagName("
                                    more_resulting_addr")
116                             if len(more_n) > 0:
117                                 for node in more_n[0].getElementsByTagName(
                                        "resulting_addr"):
118                                     t = getText(node)
119                                     if t != '':
120                                         address.append(t)
121                             dnsqueries.append([host, address])
122                         for gethostbyaddr_n in conn_n.getElementsByTagName(
                                "gethostbyaddr"):
123                             gethostbyaddr_a = gethostbyaddr_n.attributes
124                             address = getAttribute(gethostbyaddr_a, '
                                    requested_addr')
125                             host = getAttribute(gethostbyaddr_a, '
                                    resulting_host')
126                             revdnsqueries.append([address, host])
127
128                 conns_out_n = winsock_n[0].getElementsByTagName("
                        connections_outgoing")
129                 if len(conns_out_n) > 0:
130                     conn_out_n = conns_out_n[0]
131                     for conn_n in conn_out_n.getElementsByTagName("
                            connection"):
132                         transport = conn_n.attributes['transportprotocol'].
                                value
133                         remote_addr = conn_n.attributes['remoteaddr'].value
134                         remote_port = int(conn_n.attributes['remoteport'].
                                value)
135                         remote_dns = ''
136                         if conn_n.attributes['connectionestablished'].value
                                == '0':
137                             established = False
138                         else:
139                             established = True
140                         for query in dnsqueries:
141                             host = query[0]
142                             for address in query[1]:
```

```
143                                 if remote_addr == address:
144                                     remote_dns = host
145                                     break
146                                 if remote_dns != '':
147                                     break
148                             if conn_n.attributes.has_key('protocol'):
149                                 protocol = conn_n.attributes['protocol'].value
150                                 if protocol == 'IRC':
151                                     cnc = [ remote_addr, remote_port, 'irc',
                                            remote_dns ]
152
153                                     ircdata_n = conn_n.getElementsByTagName("
                                            irc_data")[0]
154                                     ircdata_a = ircdata_n.attributes
155                                     hostname = getAttribute(ircdata_a, '
                                            hostname')
156                                     servername = getAttribute(ircdata_a, '
                                            servername')
157                                     username = getAttribute(ircdata_a, '
                                            username')
158                                     realname = getAttribute(ircdata_a, '
                                            realname')
159                                     #FIXME workaround for CWsandbox parsing
                                            bug
160                                     if realname == '-Service':
161                                         realname = '-Service Pack 2'
162                                     password = getAttribute(ircdata_a, '
                                            password')
163                                     nickname = getAttribute(ircdata_a, 'nick')
164                                     if getAttribute(ircdata_a, '
                                            non_rfc_conform') == '0':
165                                         rfc_compliant = 'true'
166                                     else:
167                                         rfc_compliant = 'false'
168                                     bot_data = [hostname, realname, nickname,
                                            username, password, servername,
                                            rfc_compliant, analysis_date]
169                                     channels = []
170                                     for channel_n in ircdata_n.
                                            getElementsByTagName("channel"):
171                                         channel_a = channel_n.attributes
172                                         chan_name = getAttribute(channel_a, '
                                                name')
173                                         chan_password = getAttribute(channel_a
                                                , 'password')
174                                         chan_topic = getAttribute(channel_a, '
                                                topic_deleted')
175                                         channels.append([chan_name,
                                                chan_password, chan_topic])
176                                     notices = []
177                                     for notice_n in ircdata_n.
                                            getElementsByTagName("notice_deleted")
                                            :
178                                         notice_value = getAttribute(notice_n.
                                                attributes, 'value')
179                                         notices.append(['n', notice_value])
180                                     for priv_n in ircdata_n.
                                            getElementsByTagName("privmsg_deleted"
```

```
                                    ):
181                                     priv_value = getAttribute(priv_n.
                                            attributes, 'value')
182                                     notices.append(['p', priv_value])
183                                 malwares.append([ 'irc', cnc, bot_data,
                                        channels, notices ])
184                             elif protocol == 'HTTP':
185                                 cnc = [ remote_addr, remote_port, 'http',
                                        remote_dns ]
186                                 bot_data = [ analysis_date ]
187                                 httpcmds = []
188                                 httpdata_n = conn_n.getElementsByTagName("
                                        http_data")[0]
189                                 for httpcmd_n in httpdata_n.
                                        getElementsByTagName('http_cmd'):
190                                     httpcmd_a = httpcmd_n.attributes
191                                     url = getAttribute(httpcmd_a, 'url')
192                                     method = getAttribute(httpcmd_a, '
                                            method')
193                                     http_version = getAttribute(httpcmd_a,
                                             'http_version')
194                                     headers = []
195                                     headers_n = httpcmd_n.
                                            getElementsByTagName('header_data'
                                            )
196                                     if len(headers_n) > 0:
197                                         for header_n in headers_n[0].
                                                getElementsByTagName('header')
                                                :
198                                             t = getText(header_n)
199                                             if t != '':
200                                                 headers.append(t)
201                                     headers_s = '\n'.join(headers)
202                                     httpcmds.append([url, method,
                                            http_version, headers_s])
203                                 malwares.append(['http', cnc, bot_data,
                                        httpcmds ])
204                             else:
205                                 targets.append([ remote_addr, remote_port,
                                         transport, analysis_date, remote_dns,
                                         established ])
206                     else:
207                         targets.append([ remote_addr, remote_port,
                                transport, analysis_date, remote_dns,
                                established ])
208         ## ICMP ##
209         icmp_n = process_n.getElementsByTagName("icmp_section")
210         if len(icmp_n) > 0:
211             for ping_n in icmp_n[0].getElementsByTagName("ping"):
212                 ping_a = ping_n.attributes
213                 host = getAttribute(ping_a, 'host')
214                 remote_dns = ''
215                 for query in dnsqueries:
216                     host = query[0]
217                     for address in query[1]:
218                         if remote_addr == address:
219                             remote_dns = host
220                             break
```

```
221                     if remote_dns != '':
222                         break
223                 targets.append([ host, None, 'ICMP', analysis_date,
                        remote_dns, None ])
224     bot_id = None
225     for malware in malwares:
226         cur.execute(add_cnc, malware[1])
227         cnc_id = cur.fetchone()[0]
228         type = malware[0]
229         bot_data = malware[2]
230         bot_data.insert(0, cnc_id)
231         if type == 'irc':
232             cur.execute(add_irc_bot, bot_data)
233             bot_id = cur.fetchone()[0]
234             if len(malware[3]) > 0:
235                 channels = malware[3][0]
236                 channels.insert(0, bot_id)
237                 cur.execute(add_irc_channel, channels)
238             if len(malware[4]) > 0:
239                 notices = malware[4][0]
240                 notices.insert(0, bot_id)
241                 cur.execute(add_irc_privmsg, notices)
242             cur.execute(set_sample, [ md5, 'irc_bot', bot_id ])
243         elif type == 'http':
244             cur.execute(add_http_bot, bot_data)
245             bot_id = cur.fetchone()[0]
246             if len(malware[3]) > 0:
247                 httpcmds = malware[3][0]
248                 httpcmds.insert(0, bot_id)
249                 cur.execute(add_http_cmd, httpcmds)
250             cur.execute(set_sample, [ md5, 'http_bot', bot_id ])
251         else:
252             print "unknown malware '%s'" %(type)
253     if bot_id != None:
254         for target in targets:
255             target.insert(3, bot_id)
256             cur.execute(add_target, target)
257         cur.execute(set_processed, [ md5, 'CWSandbox' ])
258     return True
259
260 def process_report(db, report_id):
261         select = 'SELECT md5, provider, task_id, analysis_id, password,
                ' processed, XMLSERIALIZE(DOCUMENT report AS text) FROM
                ' reports WHERE id = %s'
262         selrep = 'SELECT XMLSERIALIZE(DOCUMENT report AS text) FROM
                ' reports WHERE id = %s'
263         update = 'UPDATE reports SET report = XMLPARSE(DOCUMENT %s),
                ' pcap = %s WHERE id = %s'
264         cur = db.cursor()
265         cur.execute(select, [ report_id ])
266         res = cur.fetchone()
267         if res[5] == True:
268                 # already processed
269                 print "Report %s has already been processed"
270                 return False
271         md5 = res[0]
272         provider = res[1]
273         xml = res[6]
```

```
274         if xml == None:
275                 print "Fetching %s report %s" %(provider, report_id)
276                 analysis_id = res[3]
277                 password = res[4]
278                 xml_url = "http://www.cwsandbox.org/?page=analysis&id=%
                        s&format=xml&password=%s" %(analysis_id, password)
279                 pcap_url = "http://www.cwsandbox.org/?page=download&
                        dltype=pcap&id=%s&password=%s" %(analysis_id,
                        password)
280                 http_handler = urllib2.HTTPHandler()
281                 opener = urllib2.build_opener(http_handler)
282                 # U = convert newlines
283                 fd = opener.open(xml_url, 'rU')
284                 xml = fd.read()
285                 fd.close()
286                 fd = opener.open(pcap_url, 'rb')
287                 pcap = fd.read()
288                 fd.close()
289                 # pcap is a binary format, we must escape it
290                 cur.execute(update, [ xml, psycopg2.Binary(pcap),
                        report_id ])
291                 db.commit()
292                 # we fetch from the db the serialized doc to avoid
                        parsing errors
293                 cur.execute(selrep, [ report_id ])
294                 xml = cur.fetchone()[0]
295         if provider == 'CWSandbox':
296                 print "Processing %s report %s" %(provider, report_id)
297                 val = process_cwsandbox(cur, xml)
298                 return val
299         elif provider == 'VirusTotal':
300                 print "Processing %s report %s" %(provider, report_id)
301                 val = process_virustotal(cur, xml)
302                 return val
303
304 def process_available_reports(db):
305         query = "SELECT id FROM reports WHERE available = true AND
                processed = false AND (provider = 'CWSandbox' or provider =
                'VirusTotal') ORDER BY id DESC LIMIT 100"
306         cur = db.cursor()
307         cur.execute(query)
308         count = 0
309         row = cur.fetchone()
310         while row != None:
311                 if process_report(db, row[0]):
312                         db.commit()
313                         count = count + 1
314                 row = cur.fetchone()
315         cur.close()
316         return count
317
318 if __name__ == '__main__':
319         db = psycopg2.connect (host='localhost', port='5432', database=
                'malware', user='malware_user', password='malware_password'
                )
320         cur = db.cursor()
321         if len(sys.argv) != 2:
322                 count = process_available_reports(db)
```

```
323                 print "Processed %i reports" %(count)
324         else:
325                 report_id = sys.argv[1]
326                 process_report(db, report_id)
327         db.commit()
328         cur.close()
329         db.close()
```

A.2.3 cncmap.php

Questo breve script PHP mostra la disposizione geografica dei centri di controllo utilizzando Google Maps [Google, 2008]; fa uso della classe Gmapper [Kiszka, 2008].

```
1  <?php
2          require('class.gmapper.php');
3
4          $db = NULL;
5          function db_open() {
6                  global $db;
7                  $db = pg_connect("host='localhost' port='5432' dbname='
                       malware' user='malware_user' password='
                       malware_password'");
8          }
9
10         function db_close() {
11                 global $db;
12                 pg_close($db);
13                 $db = NULL;
14         }
15
16         // db_query($query, [$param...])
17         function db_query($query) {
18                 global $db;
19                 $args = func_get_args();
20                 if(count($args) > 1) {
21                         array_shift($args); // remove $query from args
22                         return pg_query_params($db, $query, $args);
23                 } else {
24                         return pg_query($db, $query);
25                 }
26         }
27
28         $map = new gmap('google_maps_api_key');
29
30 function printcnc($id) {
31         $cnc = pg_fetch_assoc(db_query("select * from cncs_brk where id
               = $1", $id));
32         echo '<table>';
33         if($cnc['type'] == 'irc') {
34                 echo '<tr><th>id</th><th>username</th><th>password</th
                       ><th>nickname</th><th>usermode</th></tr>';
35                 $res = db_query("select * from irc_bots where cnc = $1"
                       , $id);
36                 while($row = pg_fetch_assoc($res)) {
37                         echo '<tr>';
38                         $id = $row['id'];
39                         $user = $row['username'];
```

```
40                          $pass = $row['password'];
41                          $nick = $row['nickname'];
42                          $usermode = $row['username']. ' '.$row['hostname
                                ']. ' '.$row['servername']. ' '.$row['
                                realname'];
43                          echo '</tr>';
44                  }
45          }
46          echo '</table>';
47  }
48
49  function drawMarkers() {
50          global $map;
51          static $latcount = 0, $loncount = 0, $fliplat = true;
52          $whois = 'http://'.$_SERVER['SERVER_NAME']. '/whois.php?target='
                ;
53          $res = db_query("select * from cncs_brk");
54          while($row = pg_fetch_assoc($res)) {
55                  if($row['num_bots'] == 1) {
56                          $num_bots = $row['num_bots']. ' bot';
57                  } else {
58                          $num_bots = $row['num_bots']. ' bots';
59                  }
60                  $msg =  '<b>CNC '.$row['id']. '</b><br />'.
61                          '<a href="'.$whois.$row['address']. '">'.$row['
                                address']. '</a>:'.$row['port']. '<br />'.
62                          '<a href="'.$whois.$row['dnsname']. '">'.$row['
                                dnsname']. '</a><br />'.
63                          $num_bots;
64                  $lat = 0; $lon = 0;
65                  switch($row['country']) {
66                          case 'CA':
67                          case 'DE':
68                          case 'SE':
69                                  $lat = $latcount;
70                                  $lon = $loncount;
71                                  if($flipmat) {
72                                          $latcount += 0.1;
73                                  } else {
74                                          $loncount -= 0.1;
75                                  }
76                                  $fliplat = !$fliplat;
77                          default:
78                                  $lat += $row['latitude'];
79                                  $lon += $row['longitude'];
80                  }
81                  $map->otherMarker($lat, $lon, $msg, $row['type']. '.png'
                        );
82          }
83  }
84
85  ?>
86  <html>
87  <head>
88          <title>Botnet cncs map</title>
89          <? $map->headjs(); ?>
90  </head>
91  <body onunload="GUnload()">
```

```
92          <?php $map->mapdiv('550', '850');?>
93          <?php
94                  $map->bodyjs();
95                  $map->map(2, '49.980067', '10.8731', 'normal', 0, 0, '
                        small');
96                  $map->markstart();
97                  db_open();
98                  drawMarkers();
99                  db_close();
100                 $map->markend();
101         ?>
102 </body>
103 </html>
```

A.3 Monitoraggio

I programmi raccolti in questa sezione si occupano di monitorare lo stato delle botnet archiviate in HIVE; i dati raccolti sono stati discussi nel Capitolo 5.

A.3.1 infiltrator

I sorgenti di *infiltrator* sono troppo voluminosi per poter essere inclusi in questa appendice; possono essere reperiti all'indirizzo `http://netlab-mn.unipv.it/hive`.

A.3.2 httpmole.py

Questo programma si collega alle botnet HTTP registrate in HIVE e ne acquisisce la risposta, che viene memorizzata nel database.

```
1  #!/usr/bin/python
2  import httplib
3  import psycopg2
4  import socket
5  from datetime import datetime
6  import hashlib
7  import magic
8
9  SENSOR_NAME = 'httpmole2'
10 SOCKET_TIMEOUT = 10
11
12 def connect(cur, bot, cmd):
13         query = 'SELECT address, port FROM cncs, http_bots WHERE
               http_bots.id = %s AND cncs.id = http_bots.cnc'
14         cur.execute(query, [ bot ])
15         row = cur.fetchone()
16         address = row[0]
17         port = int(row[1])
18         query = 'SELECT url, method, version, headers FROM http_cmds
               WHERE id = %s'
19         cur.execute(query, [ cmd ])
20         url, method, version, headers = cur.fetchone()
21         url = url[url.find('/'):]
22         headers_m = {}
23         for line in headers.split('\n'):
24                 k,v = line.split(':', 1)
```

```
25                  headers_m[k] = v
26          try:
27                  conn = httplib.HTTPConnection(address, port)
28                  conn.request(method, url, None, headers_m)
29                  response = conn.getresponse()
30                  rheaders = '\n'.join("%s: %s" %(k,v) for k,v in
                        response.getheaders())
31                  data = response.read()
32                  conn.close()
33                  return response.status, response.reason, rheaders, data
34          except socket.error, msg:
35                  try:
36                          return msg[0], msg[1], '', ''
37                  except:
38                          return 0, msg, '', ''
39
40  def add_sample(cur, timestamp, sensor, md5, sample):
41          query = 'SELECT md5 FROM samples WHERE md5 = %s'
42          cur.execute(query, [ md5 ])
43          if cur.rowcount == 0:
44                  query = 'INSERT INTO samples (md5, sample, size) VALUES
                        (%s, %s, %s)'
45                  cur.execute(query, [ md5, psycopg2.Binary(sample), len(
                        sample) ])
46          query = 'INSERT INTO sightings (md5,experiment,sensor) VALUES
                (%s, %s, %s)'
47          cur.execute(query, [ md5, timestamp, sensor ])
48
49  def process_bot(db, bot_id):
50          cur = db.cursor()
51          if bot_id == 65 or bot_id == 282 or bot_id == 284 or bot_id ==
                404 or bot_id == 405 or bot_id == 6904:
52                  return False
53          print "Connecting to bot %i" %(bot_id)
54          query = 'SELECT http_cmd FROM http_bots_ref where http_bot = %s
                '
55          cur.execute(query, [ bot_id ])
56          row = cur.fetchone()
57          while row != None:
58                  process_cmd(db, bot_id, row[0])
59                  row = cur.fetchone()
60          db.commit()
61          cur.close()
62
63  def process_cmd(db, bot_id, cmd_id):
64          cur = db.cursor()
65          timestamp = datetime.now().strftime('%Y%m%d %H%M%S')
66          status, reason, headers, data = connect(cur, bot_id, cmd_id)
67          print status, reason
68          hasher = hashlib.md5()
69          hasher.update(data)
70          md5 = hasher.hexdigest()
71          newval = False
72          query = 'SELECT md5(response) FROM http_replies WHERE http_bot
                = %s and http_cmd = %s'
73          cur.execute(query, [ bot_id, cmd_id ])
74          row = cur.fetchone()
75          while row != None:
```

```
76                      if row[0] != md5:
77                              newval = True
78                              break
79                      row = cur.fetchone()
80              if newval or (cur.rowcount == 0):
81                      print "Registering new response"
82                      cur.execute("INSERT INTO http_replies (timestamp,
                            http_bot, http_cmd, sensor, status, reason, headers
                            , response) VALUES ('%s', %i, %i, '%s', %i, '%s',
                            '%s', %s)" %(timestamp, bot_id, cmd_id, SENSOR_NAME
                            , status, reason, headers, psycopg2.Binary(data)))
83                      rtype = magic.buffer(data)
84                      if rtype.find('executable PE') != -1:
85                              print "Adding new sample"
86                              add_sample(cur, timestamp, SENSOR_NAME, md5,
                                    data)
87
88  if __name__ == '__main__':
89          socket.setdefaulttimeout(SOCKET_TIMEOUT)
90          magic = magic.open(magic.MAGIC_NONE)
91          magic.load()
92          db = psycopg2.connect (host='localhost', port='5432', database=
                'malware', user='malware_user', password='malware_password'
                )
93          cur = db.cursor()
94          cur.execute('SELECT id FROM http_bots')
95          count = 0
96          row = cur.fetchone()
97          while row != None:
98                  if process_bot(db, row[0]):
99                          db.commit()
100                         count = count + 1
101                 row = cur.fetchone()
102         magic.close()
103         db.commit()
104         cur.close()
105         db.close()
```

Appendice B

Schema fisico della base dati

Questo capitolo raccoglie lo schema fisico della base dati, sotto forma di DDL per domini, tipi, sequenze, tabelle, viste, funzioni e trigger.

B.1 Domini

```
1  CREATE DOMAIN md5 AS character(32)
2          CONSTRAINT md5_check CHECK ((char_length(VALUE) = 32));
3  COMMENT ON DOMAIN md5 IS 'MD5 hash in hex format';
```

B.2 Tipi

```
1  CREATE TYPE cnc_type AS ENUM (
2      'irc',
3      'http'
4  );
5  COMMENT ON TYPE cnc_type IS 'Type of command center';
6  CREATE TYPE malware_type AS ENUM (
7      'worm',
8      'irc_bot',
9      'http_bot'
10 );
11 COMMENT ON TYPE malware_type IS 'Malware type categorization';
12 CREATE TYPE provider AS ENUM (
13     'Anubis',
14     'CWSandbox',
15     'VirusTotal'
16 );
17 COMMENT ON TYPE provider IS 'Analysis report provider';
```

B.3 Sequenze

Le sequenze sono i contatori autoincrementanti usati per i campi di tipo `serial`.

```
1  CREATE SEQUENCE av_reports_id_seq;
2  CREATE SEQUENCE cncs_id_seq;
3  CREATE SEQUENCE http_cmds_id_seq;
4  CREATE SEQUENCE http_replies_id_seq;
5  CREATE SEQUENCE irc_channels_id_seq;
6  CREATE SEQUENCE irc_commands_id_seq;
7  CREATE SEQUENCE irc_notices_id_seq;
```

```
8  CREATE SEQUENCE malware_id_seq;
9  COMMENT ON SEQUENCE malware_id_seq IS 'Malware ID; used in worms and *
       _bots, referenced in samples.';
10 CREATE SEQUENCE reports_id_seq;
11 CREATE SEQUENCE sightings_id_seq;
12 CREATE SEQUENCE targets_id_seq;
```

B.4 Tabelle

B.4.1 av_reports

```
1  CREATE TABLE av_reports
2  (
3          id serial NOT NULL,
4          md5 md5 NOT NULL,
5          scanner varchar(20) NOT NULL,
6          scanner_version varchar(15),
7          signature_version date,
8          classification varchar(100) NOT NULL,
9          processed boolean NOT NULL DEFAULT false,
10         CONSTRAINT "antivirus_reports_pkey" PRIMARY KEY (id)
11 );
12 -- Indexes
13 CREATE UNIQUE INDEX antivirus_reports_uc2 ON av_reports USING btree (
       md5, scanner);
14 CREATE INDEX av_reports_idx3 ON av_reports USING btree (md5);
```

B.4.2 cncs

```
1  CREATE TABLE cncs
2  (
3          id serial NOT NULL,
4          address inet NOT NULL,
5          port integer NOT NULL,
6          type cnc_type NOT NULL,
7          dnsname varchar(50),
8          country char(2),
9          active boolean,
10         latitude double precision,
11         longitude double precision,
12         CONSTRAINT "cnc_pkey" PRIMARY KEY (id)
13 );
14 -- Indexes
15 CREATE UNIQUE INDEX cnc_uc2 ON cncs USING btree (address, port);
16 -- Comments
17 COMMENT ON COLUMN public.cncs.type IS 'irc or http';
18 COMMENT ON COLUMN public.cncs.dnsname IS 'DNS name as requested by the
       malware';
19 COMMENT ON COLUMN public.cncs.country IS 'ISO 3166 country code';
20 -- Triggers
21 CREATE TRIGGER set_geo BEFORE INSERT ON cncs FOR EACH ROW EXECUTE
       PROCEDURE set_geo();
```

B.4.3 http_bots

```
1  CREATE TABLE http_bots
2  (
3          id serial NOT NULL,
4          cnc integer NOT NULL,
5          last_seen timestamp,
6          CONSTRAINT "http_bots_pkey" PRIMARY KEY (id),
7          CONSTRAINT "http_bots_cnc" FOREIGN KEY (cnc) REFERENCES cncs(id
               )
8  );
9  -- Indexes
10 CREATE UNIQUE INDEX http_bots_uc3 ON http_bots USING btree (cnc);
```

B.4.4 http_bots_ref

```
1  CREATE TABLE http_bots_ref
2  (
3          http_bot integer NOT NULL,
4          http_cmd integer NOT NULL,
5          CONSTRAINT "http_bots_ref_pkey" PRIMARY KEY (http_bot, http_cmd
               ),
6          CONSTRAINT "http_bots_ref_bot" FOREIGN KEY (http_bot)
               REFERENCES http_bots(id),
7          CONSTRAINT "http_bots_ref_cmd" FOREIGN KEY (http_cmd)
               REFERENCES http_cmds(id)
8  );
```

B.4.5 http_cmds

```
1  CREATE TABLE http_cmds
2  (
3          id serial NOT NULL,
4          url varchar(500) NOT NULL,
5          method varchar(5) NOT NULL,
6          version varchar(10) NOT NULL,
7          headers text,
8          CONSTRAINT "http_cmds_pkey" PRIMARY KEY (id)
9  );
```

B.4.6 http_replies

```
1  CREATE TABLE http_replies
2  (
3          id serial NOT NULL,
4          "timestamp" timestamp NOT NULL,
5          http_bot integer NOT NULL,
6          http_cmd integer NOT NULL,
7          status smallint NOT NULL,
8          reason varchar(30) NOT NULL,
9          headers text,
10         response bytea,
11         sensor varchar(10),
12         CONSTRAINT "http_replies_pkey" PRIMARY KEY (id),
13         CONSTRAINT "http_replies_http_bot" FOREIGN KEY (http_bot)
               REFERENCES http_bots(id) ON UPDATE CASCADE ON DELETE
               CASCADE,
14         CONSTRAINT "http_replies_http_cmd" FOREIGN KEY (http_cmd)
               REFERENCES http_cmds(id) ON UPDATE CASCADE ON DELETE
               CASCADE,
```

```
15          CONSTRAINT "http_replies_sensor" FOREIGN KEY (sensor)
                REFERENCES sensors(name)
16  );
```

B.4.7 irc_bots

```
1   CREATE TABLE irc_bots
2   (
3           id serial NOT NULL,
4           cnc integer NOT NULL,
5           hostname varchar(20),
6           realname varchar(400),
7           nickname varchar(50),
8           username varchar(20),
9           password varchar(20),
10          rfc_compliant boolean,
11          servername varchar(20),
12          last_seen timestamp,
13          CONSTRAINT "irc_bot_pkey" PRIMARY KEY (id),
14          CONSTRAINT "irc_bot_cnc" FOREIGN KEY (cnc) REFERENCES cncs(id)
15  );
```

B.4.8 irc_bots_refc

```
1   CREATE TABLE irc_bots_refc
2   (
3           irc_bot integer NOT NULL,
4           irc_channel integer NOT NULL,
5           CONSTRAINT "irc_bots_ref_pkey" PRIMARY KEY (irc_bot,
                irc_channel),
6           CONSTRAINT "irc_bots_ref_bot" FOREIGN KEY (irc_bot) REFERENCES
                irc_bots(id),
7           CONSTRAINT "irc_bots_ref_channel" FOREIGN KEY (irc_channel)
                REFERENCES irc_channels(id)
8   );
```

B.4.9 irc_bots_refp

```
1   CREATE TABLE irc_bots_refp
2   (
3           irc_bot integer NOT NULL,
4           irc_privmsg integer NOT NULL,
5           CONSTRAINT "irc_bots_refp_pkey" PRIMARY KEY (irc_bot,
                irc_privmsg),
6           CONSTRAINT "irc_bots_refp_bot" FOREIGN KEY (irc_bot) REFERENCES
                 irc_bots(id),
7           CONSTRAINT "irc_bots_refp_privmsg" FOREIGN KEY (irc_privmsg)
                REFERENCES irc_privmsgs(id)
8   );
```

B.4.10 irc_channels

```
1   CREATE TABLE irc_channels
2   (
3           id serial NOT NULL,
4           name varchar(15) NOT NULL,
5           password varchar(15),
```

```
6           topics varchar(500),
7           CONSTRAINT "irc_channels_pkey" PRIMARY KEY (id)
8   );
```

B.4.11 irc_commands

```
1   CREATE TABLE irc_commands
2   (
3           id serial NOT NULL,
4           irc_bot integer NOT NULL,
5           irc_channel integer,
6           "timestamp" timestamp NOT NULL,
7           command text NOT NULL,
8           sensor varchar(15),
9           CONSTRAINT "irc_commands_pkey" PRIMARY KEY (id),
10          CONSTRAINT "irc_commands_sensor" FOREIGN KEY (sensor)
                REFERENCES sensors(name),
11          CONSTRAINT "irc_commands_bot" FOREIGN KEY (irc_bot) REFERENCES
                irc_bots(id),
12          CONSTRAINT "irc_commands_channel" FOREIGN KEY (irc_channel)
                REFERENCES irc_channels(id)
13  );
```

B.4.12 irc_privmsgs

```
1   CREATE TABLE irc_privmsgs
2   (
3           id serial NOT NULL,
4           type char(1) NOT NULL,
5           message varchar(500) NOT NULL,
6           CONSTRAINT "irc_notices_pkey" PRIMARY KEY (id)
7   );
8
9   -- Comments
10  COMMENT ON COLUMN irc_privmsgs.type IS 'Notice or Privmsg';
```

B.4.13 reports

```
1   CREATE TABLE reports
2   (
3           id serial NOT NULL,
4           md5 md5 NOT NULL,
5           provider provider NOT NULL,
6           available boolean DEFAULT false,
7           task_id char(32),
8           analysis_id integer,
9           password char(5),
10          processed boolean DEFAULT false,
11          report xml,
12          pcap bytea,
13          CONSTRAINT "reports_pkey" PRIMARY KEY (id),
14          CONSTRAINT "reports_md5" FOREIGN KEY (md5) REFERENCES samples(
                md5)
15  );
16  -- Indexes
17  CREATE UNIQUE INDEX reports_uc6 ON reports USING btree (task_id);
18  CREATE UNIQUE INDEX reports_uc7 ON reports USING btree (analysis_id);
```

```
19 CREATE UNIQUE INDEX reports_uc5 ON reports USING btree (md5, provider);
20 CREATE INDEX reports_idx_md5 ON reports USING btree (md5);
21 -- Comments
22 COMMENT ON COLUMN reports.report IS 'Report data in XML format (if
       available)';
23 COMMENT ON COLUMN reports.pcap IS 'PCAP network trace (if available)';
```

B.4.14 samples

```
1  CREATE TABLE samples
2  (
3          md5 md5 NOT NULL,
4          size integer,
5          sample bytea,
6          CONSTRAINT "malware_pkey" PRIMARY KEY (md5)
7  );
8  -- Comments
9  COMMENT ON COLUMN samples.md5 IS 'MD5 hash of the collected sample';
10 COMMENT ON COLUMN samples.size IS 'Sample size in bytes';
11 COMMENT ON COLUMN samples.sample IS 'The collected sample binary file';
12 -- Triggers
13 CREATE TRIGGER check_sample_file BEFORE INSERT OR DELETE OR UPDATE ON
       samples FOR EACH ROW EXECUTE PROCEDURE check_sample_file();
14 CREATE TRIGGER do_submit AFTER INSERT ON samples FOR EACH ROW EXECUTE
       PROCEDURE do_submit();
```

B.4.15 samples_ref

```
1  CREATE TABLE samples_ref
2  (
3          sample md5 NOT NULL,
4          type malware_type NOT NULL,
5          malware_id integer NOT NULL,
6          CONSTRAINT "samples_ref_pk" PRIMARY KEY (sample, malware_id)
7  );
8  -- Comments
9  COMMENT ON COLUMN samples_ref.sample IS 'Sample MD5 hash';
10 COMMENT ON COLUMN samples_ref.type IS 'Malware type';
11 COMMENT ON COLUMN samples_ref.malware_id IS 'Malware identification for
       this sample; references to <type>  table''s field ''id''';
```

B.4.16 sensors

```
1  CREATE TABLE sensors
2  (
3          name varchar(15) NOT NULL,
4          "os" varchar(20),
5          ip inet,
6          low_interaction boolean DEFAULT false,
7          notes varchar(200),
8          CONSTRAINT "sensors_pkey" PRIMARY KEY (name)
9  );
```

B.4.17 settings

```
1  CREATE TABLE settings
2  (
```

```
3           key varchar(20) NOT NULL,
4           value text NOT NULL,
5           CONSTRAINT "settings_pkey" PRIMARY KEY (key)
6   );
```

B.4.18 sightings

```
1   CREATE TABLE public.sightings
2   (
3           id serial NOT NULL,
4           md5 char(32) NOT NULL,
5           sensor varchar(10) NOT NULL,
6           experiment timestamp,
7           filename varchar(255),
8           filepath varchar(255),
9           filedate timestamp,
10          CONSTRAINT "sightings_pkey" PRIMARY KEY (id),
11          CONSTRAINT "sightings_sensor" FOREIGN KEY (sensor) REFERENCES
                sensors(name),
12          CONSTRAINT "sightings_md5" FOREIGN KEY (md5) REFERENCES samples
                (md5) ON UPDATE CASCADE ON DELETE CASCADE
13  );
```

B.4.19 targets

```
1   CREATE TABLE targets
2   (
3           id serial NOT NULL,
4           address inet NOT NULL,
5           port integer,
6           protocol varchar(5) NOT NULL,
7           attacker integer NOT NULL,
8           last_seen timestamp NOT NULL,
9           dnsname varchar(50),
10          established boolean,
11          CONSTRAINT "targets_pkey" PRIMARY KEY (id)
12  );
```

B.4.20 worms

```
1   CREATE TABLE worms
2   (
3           id serial DEFAULT nextval('malware_id_seq'::regclass) NOT NULL,
4           name varchar(10) NOT NULL,
5           CONSTRAINT "worms_pkey" PRIMARY KEY (id)
6   );
```

B.5 Viste

B.5.1 av_accuracy

```
1   CREATE VIEW av_accuracy AS
2       SELECT
3           a.scanner,
4           (((a.ok_samples)::double precision / (b.scanned_samples)::
                double precision) * (100)::double precision) AS
                false_negative_percentage
```

```
5       FROM (
6           (SELECT av_reports.scanner, count(av_reports.md5) AS ok_samples
                FROM av_reports WHERE ((av_reports.classification)::text =
                'OK'::text) GROUP BY av_reports.scanner) a
7           NATURAL JOIN
8           (SELECT av_reports.scanner, count(av_reports.md5) AS
                scanned_samples FROM av_reports GROUP BY av_reports.scanner
                ) b
9       );
```

B.5.2 cncs_brk

```
1   CREATE VIEW cncs_brk AS
2       SELECT
3           cncs.id, cncs.address, cncs.port, cncs.dnsname, cncs.type,
                count(http_bots.id) AS num_bots, max(http_bots.last_seen)
                AS last_seen, cncs.country, cncs.latitude, cncs.longitude
4       FROM http_bots, cncs
5       WHERE (http_bots.cnc = cncs.id)
6       GROUP BY cncs.id, cncs.address, cncs.port, cncs.dnsname, cncs.type,
            cncs.country, cncs.latitude, cncs.longitude
7       UNION SELECT
8           cncs.id, cncs.address, cncs.port, cncs.dnsname, cncs.type,
                count(irc_bots.id) AS num_bots, max(irc_bots.last_seen) AS
                last_seen, cncs.country, cncs.latitude, cncs.longitude
9       FROM irc_bots, cncs
10      WHERE (irc_bots.cnc = cncs.id)
11      GROUP BY cncs.id, cncs.address, cncs.port, cncs.dnsname, cncs.type,
            cncs.country, cncs.latitude, cncs.longitude
12      ORDER BY num_bots DESC;
```

B.5.3 daily_samples_per_type

```
1   CREATE VIEW daily_samples_per_type AS
2       SELECT
3           sub.date,
4           sum(CASE WHEN (sub.type = 'worm'::malware_type) THEN 1 ELSE 0
                END) AS worms,
5           sum(CASE WHEN (sub.type = 'irc_bot'::malware_type) THEN 1 ELSE
                0 END) AS irc_bots,
6           sum(CASE WHEN (sub.type = 'http_bot'::malware_type) THEN 1 ELSE
                0 END) AS http_bots,
7           count(sub.type) AS total
8       FROM (
9           SELECT DISTINCT ON (sightings.md5)
10                  (sightings.experiment)::date AS date, samples_ref.type
11          FROM samples_ref, sightings
12          WHERE ((sightings.md5)::bpchar = (samples_ref.sample)::bpchar)
13          ORDER BY sightings.md5
14      ) sub
15      GROUP BY sub.date ORDER BY sub.date;
```

B.5.4 daily_sightings_per_type

```
1   CREATE VIEW daily_sightings_per_type AS
2       SELECT
3           (sightings.experiment)::date AS date,
```

```
4           sum(CASE WHEN (samples_ref.type = 'worm'::malware_type) THEN 1
                ELSE 0 END) AS worms,
5           sum(CASE WHEN (samples_ref.type = 'irc_bot'::malware_type) THEN
                 1 ELSE 0 END) AS irc_bots,
6           sum(CASE WHEN (samples_ref.type = 'http_bot'::malware_type)
                THEN 1 ELSE 0 END) AS http_bots,
7           count(samples_ref.type) AS total
8       FROM sightings, samples_ref
9       WHERE ((sightings.md5)::bpchar = (samples_ref.sample)::bpchar)
10      GROUP BY (sightings.experiment)::date ORDER BY (sightings.
            experiment)::date;
```

B.5.5 irc_bots_viruses

```
1   CREATE VIEW irc_bots_viruses AS
2       SELECT samples.md5, samples_ref.malware_id, cncs.address, cncs.
            dnsname, av_reports.classification
3       FROM samples, samples_ref, av_reports, irc_bots, cncs
4       WHERE samples.md5::bpchar = samples_ref.sample::bpchar AND
            samples_ref.malware_id = irc_bots.id AND samples.md5::bpchar =
            av_reports.md5::bpchar AND av_reports.scanner::text = 'ClamAV-
            local'::text AND irc_bots.cnc = cncs.id
5       ORDER BY av_reports.classification;
```

B.5.6 irc_commands_brk

```
1   CREATE VIEW irc_commands_brk AS
2       SELECT
3           sum(CASE WHEN (irc_commands.command ~~ '%PRIVMSG%'::text) THEN
                1 ELSE 0 END) AS priv,
4           sum(CASE WHEN (irc_commands.command ~~ '%NOTICE%'::text) THEN 1
                 ELSE 0 END) AS notice,
5           sum(CASE WHEN (irc_commands.command ~~ '%QUIT%'::text) THEN 1
                ELSE 0 END) AS quit,
6           sum(CASE WHEN (irc_commands.command ~~ '%JOIN%'::text) THEN 1
                ELSE 0 END) AS "join",
7           sum(CASE WHEN (irc_commands.command ~~ '%MODE%'::text) THEN 1
                ELSE 0 END) AS mode,
8           sum(CASE WHEN (irc_commands.command ~~ '%TOPIC%'::text) THEN 1
                ELSE 0 END) AS topic,
9           sum(CASE WHEN (lower(irc_commands.command) ~~ '%scan%'::text OR
                 lower(irc_commands.command) ~~ '%.asc asn%'::text) THEN 1
                ELSE 0 END) AS bot_scan,
10          sum(CASE WHEN (lower(irc_commands.command) ~~ '%!login%'::text)
                 THEN 1 ELSE 0 END) AS bot_login,
11          sum(CASE WHEN (lower(irc_commands.command) ~~ '%!get%'::text OR
                 lower(irc_commands.command) ~~ '%.get%'::text OR lower(
                irc_commands.command) ~~ '%*get%'::text OR lower(
                irc_commands.command) ~~ '%!nokia%'::text) THEN 1 ELSE 0
                END) AS bot_update,
12          sum(CASE WHEN (lower(irc_commands.command) ~~ '%*pst %'::text)
                THEN 1 ELSE 0 END) AS data_mining
13      FROM irc_commands;
```

B.5.7 path_brk

```
1   CREATE VIEW path_brk AS
```

```
2       SELECT sightings.filepath, count(sightings.filepath) AS
            num_sightings FROM sightings GROUP BY sightings.filepath ORDER
            BY count(sightings.filepath) DESC;
```

B.5.8 samples_per_day

```
1  CREATE VIEW samples_per_day AS
2       SELECT
3           sum(pippo2.num_samples) AS num_samples, pippo2.experiment
4       FROM (
5           SELECT
6                   count(pippo.md5) AS num_samples, (pippo.experiment)::
                        date AS experiment
7           FROM (
8                   SELECT DISTINCT ON (samples.md5) samples.md5, sightings
                        .experiment
9                   FROM samples, sightings
10                  WHERE ((samples.md5)::bpchar = (sightings.md5)::bpchar)
                        ORDER BY samples.md5
11          ) pippo
12          GROUP BY pippo.experiment
13      ) pippo2
14      GROUP BY pippo2.experiment ORDER BY pippo2.experiment;
```

B.5.9 sensors_brk

```
1  CREATE VIEW sensors_brk AS
2       SELECT sightings.sensor, count(*) AS num_sightings FROM sightings
            GROUP BY sightings.sensor ORDER BY count(*) DESC;
```

B.5.10 sightings_brk

```
1  CREATE VIEW sightings_brk AS
2       SELECT sightings.md5, count(*) AS num_sightings FROM samples,
            sightings WHERE ((samples.md5)::bpchar = (sightings.md5)::
            bpchar) GROUP BY sightings.md5 ORDER BY count(*) DESC;
```

B.5.11 sightings_per_day

```
1  CREATE VIEW sightings_per_day AS
2       SELECT sum(pippo2.num_sightings) AS num_sightings, pippo2.
            experiment FROM (SELECT count(sightings.md5) AS num_sightings,
            (sightings.experiment)::date AS experiment FROM sightings GROUP
             BY sightings.experiment) pippo2 GROUP BY pippo2.experiment
            ORDER BY pippo2.experiment;
```

B.5.12 size_brk

```
1  CREATE VIEW size_brk AS
2       SELECT samples.size, count(samples.size) AS num_samples FROM
            samples GROUP BY samples.size ORDER BY count(samples.size) DESC
            ;
```

B.5.13 type_brk

```
1  CREATE VIEW type_brk AS
2      SELECT '(unfiled)' AS type, count(samples.md5) AS num_samples, NULL
           ::unknown AS num_unique FROM samples
3                  WHERE (NOT ((samples.md5)::bpchar IN (
4                          SELECT DISTINCT samples_ref.sample FROM
                               samples_ref ORDER BY samples_ref.sample))
5                   )
6      UNION SELECT (samples_ref.type)::text AS type, count(samples_ref.
           sample) AS num_samples, count(DISTINCT irc_bots.cnc) AS
           num_unique
7                  FROM samples_ref, irc_bots
8                  WHERE (samples_ref.malware_id = irc_bots.id) GROUP BY
                       samples_ref.type)
9      UNION SELECT (samples_ref.type)::text AS type, count(samples_ref.
           sample) AS num_samples, count(DISTINCT http_bots.cnc) AS
           num_unique
10         FROM samples_ref, http_bots
11         WHERE (samples_ref.malware_id = http_bots.id) GROUP BY
               samples_ref.type)
12     UNION SELECT (samples_ref.type)::text AS type, count(samples_ref.
           sample) AS num_samples, count(DISTINCT worms.id) AS num_unique
13         FROM samples_ref, worms
14         WHERE (samples_ref.malware_id = worms.id) GROUP BY samples_ref.
               type;
```

B.5.14 weekly_samples_per_type

```
1  CREATE VIEW weekly_samples_per_type AS
2      SELECT
3          date_part('week'::text, daily_samples_per_type.date) AS week,
4          sum(daily_samples_per_type.worms) AS worms,
5          sum(daily_samples_per_type.irc_bots) AS irc_bots,
6          sum(daily_samples_per_type.http_bots) AS http_bots,
7          sum(daily_samples_per_type.total) AS total
8      FROM daily_samples_per_type GROUP BY week ORDER BY week;
```

B.5.15 weekly_sightings_per_type

```
1  CREATE VIEW weekly_sightings_per_type AS
2      SELECT
3          date_part('week'::text, daily_sightings_per_type.date) AS week,
4          sum(daily_sightings_per_type.worms) AS worms,
5          sum(daily_sightings_per_type.irc_bots) AS irc_bots,
6          sum(daily_sightings_per_type.http_bots) AS http_bots,
7          sum(daily_sightings_per_type.total) AS total
8      FROM daily_sightings_per_type GROUP BY week ORDER BY week;
```

B.6 Funzioni

B.6.1 add_cnc

```
1  CREATE or REPLACE FUNCTION "public"."add_cnc"(
2  IN "myaddress" inet,
3  IN "myport" int4,
4  IN "mytype" cnc_type,
5  IN "mydnsname" varchar)
6  RETURNS "pg_catalog"."int4" AS
```

```
7  $BODY$
8  DECLARE
9          val integer;
10         dns varchar;
11 BEGIN
12       SELECT id, dnsname INTO val, dns FROM cncs
13              WHERE address = myaddress and port = myport and type =
                    mytype;
14       IF val IS NULL THEN
15            INSERT INTO cncs (id, address, port, type, dnsname)
16                   VALUES (DEFAULT, myaddress, myport, mytype, mydnsname)
17                   RETURNING id INTO val;
18       ELSEIF dns IS NULL AND mydnsname IS NOT NULL THEN
19            UPDATE cncs SET dnsname = mydnsname WHERE id = val;
20       END IF;
21       RETURN val;
22 END;
23 $BODY$
24 LANGUAGE 'plpgsql' VOLATILE;
```

B.6.2 add_http_bot

```
1  CREATE or REPLACE FUNCTION "public"."add_http_bot"(
2  IN "mycnc" int4,
3  IN "mydate" timestamp)
4  RETURNS "pg_catalog"."int4" AS
5  $BODY$
6  DECLARE
7          val integer;
8          olddate timestamp;
9  BEGIN
10       SELECT id, last_seen INTO val, olddate FROM http_bots WHERE cnc =
               mycnc;
11       IF val IS NULL THEN
12          INSERT INTO http_bots (id, cnc, last_seen)
13                 VALUES (DEFAULT, mycnc, mydate) RETURNING id INTO val;
14       ELSEIF mydate > olddate THEN
15          UPDATE http_bots SET last_seen = mydate WHERE id = val;
16       END IF;
17       RETURN val;
18 END;
19 $BODY$
20 LANGUAGE 'plpgsql' VOLATILE;
```

B.6.3 add_http_cmd

```
1  CREATE or REPLACE FUNCTION "public"."add_http_cmd"(
2  IN "mybot" int4,
3  IN "myurl" varchar,
4  IN "mymethod" varchar,
5  IN "myversion" varchar,
6  IN "myheaders" text)
7  RETURNS "pg_catalog"."int4" AS
8  $BODY$
9  DECLARE
10         val integer;
11         temprow http_bots_ref%ROWTYPE;
12 BEGIN
```

```
13      SELECT id INTO val FROM http_cmds WHERE
14              url = myurl AND method = mymethod AND
15              version = myversion AND headers = myheaders;
16      IF val IS NULL THEN
17         INSERT INTO http_cmds (id, url, method, version, headers)
18              VALUES (DEFAULT, myurl, mymethod, myversion, myheaders)
19              RETURNING id INTO val;
20      END IF;
21      SELECT * INTO temprow FROM http_bots_ref WHERE http_bot = mybot
            AND http_cmd = val;
22      IF NOT FOUND THEN
23           INSERT INTO http_bots_ref (http_bot, http_cmd) VALUES (mybot,
                 val);
24      ELSE
25           RAISE NOTICE 'Command␣%␣is␣already␣linked␣to␣bot␣%,␣skipping'
                 , val, mybot;
26      END IF;
27      RETURN val;
28 END;
29 $BODY$
30 LANGUAGE 'plpgsql' VOLATILE;
31 COMMENT ON FUNCTION add_http_cmd(mybot integer, myurl character varying
       , mymethod character varying, myversion character varying,
       myheaders text) IS 'Add␣a␣new␣HTTP␣command␣and␣link␣it␣with␣the␣
       provided␣bot.␣If␣an␣identical␣command␣already␣exists␣only␣establish
       ␣the␣link.␣If␣an␣identical␣link␣already␣exists␣raise␣a␣notice␣and␣
       abort.';
```

B.6.4 add_irc_bot

```
1  CREATE FUNCTION add_irc_bot(mycnc integer, myhostname character varying
       , myrealname character varying, mynickname character varying,
       myusername character varying, mypassword character varying,
       myservername character varying, myrfccomp boolean, mydate timestamp
        without time zone) RETURNS integer
2      AS $$
3  DECLARE
4          val integer;
5          olddate timestamp;
6  BEGIN
7       SELECT id, last_seen INTO val, olddate FROM irc_bots WHERE cnc =
            mycnc AND
8               hostname = myhostname AND realname = myrealname AND
9               nickname = mynickname AND username = myusername AND
10              password = mypassword AND servername = myservername AND
11              rfc_compliant = myrfccomp;
12      IF val IS NULL THEN
13              INSERT INTO irc_bots
14                      (id,cnc,hostname,realname,nickname,username,password
                            ,servername,rfc_compliant,last_seen)
15              VALUES (DEFAULT,mycnc,myhostname,myrealname,mynickname,
                    myusername,mypassword,myservername,myrfccomp,mydate)
16              RETURNING id INTO val;
17      ELSEIF mydate > olddate THEN
18              UPDATE irc_bots SET last_seen = mydate WHERE id = val;
19      END IF;
20      RETURN val;
21 END;
```

```
22 $$
23     LANGUAGE plpgsql;
```

B.6.5 add_irc_channel

```
1  CREATE FUNCTION add_irc_channel(mybot integer, myname character varying
       , mypassword character varying, mytopics text) RETURNS integer
2      AS $$
3  DECLARE
4          val integer;
5          myrow irc_channels%ROWTYPE;
6          temprow irc_bots_refc%ROWTYPE;
7  BEGIN
8       SELECT id INTO val FROM irc_channels WHERE
9              name = myname AND password = mypassword AND topics =
                   mytopics;
10      IF val IS NULL THEN
11             SELECT * INTO myrow FROM irc_channels WHERE name = myname;
12             val := myrow.id;
13             IF val IS NULL OR NOT (
14                 (myrow.name = myname) AND
15                 (myrow.password IS NULL OR myrow.password = mypassword)
                       AND
16                 (myrow.topics IS NULL OR myrow.topics = mytopics)
17             ) THEN
18                     INSERT INTO irc_channels (id, name, password, topics
                           )
19                         VALUES (DEFAULT, myname, mypassword, mytopics
                               )
20                         RETURNING id INTO val;
21             END IF;
22      END IF;
23      SELECT * INTO temprow FROM irc_bots_refc WHERE irc_bot = mybot AND
            irc_channel = val;
24      IF NOT FOUND THEN
25             INSERT INTO irc_bots_refc (irc_bot, irc_channel) VALUES (
                   mybot, val);
26      ELSE
27             RAISE NOTICE 'Channel␣%␣is␣already␣linked␣to␣bot␣%,␣skipping'
                   , val, mybot;
28      END IF;
29      RETURN val;
30 END;
31 $$
32     LANGUAGE plpgsql;
33 COMMENT ON FUNCTION add_irc_channel(mybot integer, myname character
       varying, mypassword character varying, mytopics text) IS 'Add␣a␣new
       ␣IRC␣channel␣and␣link␣it␣with␣the␣provided␣bot.␣If␣an␣identical␣
       channel␣already␣exists␣only␣establish␣the␣link.␣If␣an␣identical␣
       link␣already␣exists␣raise␣a␣notice␣and␣abort.';
```

B.6.6 add_irc_privmsg

```
1  CREATE FUNCTION add_irc_privmsg(mybot integer, mytype character varying
       , mymessage character varying) RETURNS integer
2      AS $$
3  DECLARE
4          val integer;
```

```
5         temprow irc_bots_refp%ROWTYPE;
6  BEGIN
7       SELECT id INTO val FROM irc_privmsgs WHERE
8               type = mytype AND message = mymessage;
9       IF val IS NULL THEN
10              INSERT INTO irc_privmsgs (id, type, message) VALUES (
                    DEFAULT, mytype, mymessage)
11              RETURNING id INTO val;
12      END IF;
13      SELECT * INTO temprow FROM irc_bots_refp WHERE irc_bot = mybot AND
            irc_privmsg = val;
14      IF NOT FOUND THEN
15          INSERT INTO irc_bots_refp (irc_bot, irc_privmsg) VALUES (
                mybot, val);
16      ELSE
17          RAISE NOTICE 'Message␣%␣is␣already␣linked␣to␣bot␣%,␣skipping'
                , val, mybot;
18      END IF;
19      RETURN val;
20 END;
21 $$
22      LANGUAGE plpgsql;
23 COMMENT ON FUNCTION add_irc_privmsg(mybot integer, mytype character
       varying, mymessage character varying) IS 'Add␣a␣new␣IRC␣message␣and
       ␣link␣it␣with␣the␣provided␣bot.␣If␣an␣identical␣message␣already␣
       exists␣only␣establish␣the␣link.␣If␣an␣identical␣link␣already␣exists
       ␣raise␣a␣notice␣and␣abort.';
```

B.6.7 add_target

```
1  CREATE FUNCTION add_target(myaddress inet, myport integer, myproto
       character varying, myattacker integer, mydate timestamp without
       time zone, mydnsname character varying, myestablished boolean)
       RETURNS integer
2       AS $$
3  DECLARE
4          val integer;
5          olddate timestamp;
6  BEGIN
7       SELECT id, last_seen INTO val, olddate FROM targets
8               WHERE address = myaddress AND (port = myport OR (port IS
                    NULL AND protocol = 'ICMP'))
9                       AND protocol = myproto AND attacker = myattacker AND
                            dnsname = mydnsname;
10      IF val IS NULL THEN
11          INSERT INTO targets (id, address, port, protocol, attacker,
                last_seen, dnsname, established)
12                  VALUES (DEFAULT, myaddress, myport, myproto, myattacker,
                            mydate, mydnsname, myestablished)
13                  RETURNING id INTO val;
14      ELSEIF mydate > olddate THEN
15          UPDATE targets SET last_seen = mydate WHERE id = val;
16      END IF;
17      RETURN val;
18 END;
19 $$
20      LANGUAGE plpgsql;
```

B.6.8 process_av_report

```
1  CREATE FUNCTION process_av_report(mymd5 character) RETURNS boolean
2      AS $$
3  DECLARE
4          -- constants
5          WORM_TYPE CONSTANT malware_type DEFAULT 'worm';
6          ALLAPLE_ID CONSTANT integer DEFAULT 1;
7          ALLAPLE_THRESHOLD CONSTANT integer DEFAULT 5;
8          -- variables
9          myrow av_reports%ROWTYPE;
10         classification varchar(100);
11         val boolean DEFAULT false;
12         allaple integer DEFAULT 0;
13 BEGIN
14       FOR myrow IN SELECT * FROM av_reports WHERE md5 = mymd5 LOOP
15           classification := lower(myrow.classification);
16           IF classification LIKE '%allaple%' THEN
17               allaple := allaple + 1;
18           END IF;
19       END LOOP;
20       IF allaple > ALLAPLE_THRESHOLD THEN
21           RAISE NOTICE 'Sample␣%␣is␣ALLAPLE,␣updating.', mymd5;
22           SELECT set_sample(mymd5, WORM_TYPE, ALLAPLE_ID) INTO val;
23           UPDATE av_reports SET processed = true WHERE md5 = mymd5;
24       END IF;
25       RETURN val;
26 END;
27 $$
28     LANGUAGE plpgsql;
```

B.6.9 set_sample

```
1  CREATE FUNCTION set_sample(mysample character, mytype malware_type,
       mymalware_id integer) RETURNS boolean
2      AS $$
3  DECLARE
4          temprow RECORD;
5  BEGIN
6        SELECT * INTO temprow FROM samples_ref WHERE sample = mysample AND
             malware_id = mymalware_id;
7        IF NOT FOUND THEN
8            INSERT INTO samples_ref (sample, type, malware_id) VALUES (
                 mysample, mytype, mymalware_id);
9            RETURN TRUE;
10       ELSE
11           RAISE WARNING 'Malware␣%␣(type␣%)␣is␣already␣linked␣to␣sample␣
                 %,␣skipping', mymalware_id, mytype, mysample;
12           RETURN FALSE;
13       END IF;
14 END;
15 $$
16     LANGUAGE plpgsql;
```

B.6.10 submit_to_anubis

```
1  CREATE FUNCTION submit_to_anubis(md5 character) RETURNS boolean
```

```
2       AS $_$
3   import sys
4   sys.path.append('/home/davide')
5   import httputils
6   from pg import unescape_bytea
7
8   if not SD.has_key('selcheck'):
9       SD['selcheck'] = plpy.prepare(
10          "SELECT md5 FROM reports WHERE md5 = $1 and provider = 'Anubis'
                ",
11          [ 'text' ])
12
13  if len(plpy.execute(SD['selcheck'], [ md5 ])) > 0:
14      return 'false'
15
16  if not SD.has_key('submit_url'):
17      res = plpy.execute("select value as submit_url from settings where
            key = 'ANUBIS_SUBMIT_URL'")
18      SD['submit_url'] = res[0]['submit_url']
19
20  if not SD.has_key('sample_check'):
21      SD['sample_check'] = plpy.prepare('SELECT sample FROM samples WHERE
            md5 = $1 AND sample IS NOT NULL', [ "text" ])
22
23  res = plpy.execute(SD['sample_check'], [ md5 ])
24  if len(res) > 0:
25      file = md5
26      content = unescape_bytea(res[0]['sample'])
27  else:
28      file = "/var/www/uploads/%s" %(md5)
29      fd = open(file, "rb")
30      content = fd.read()
31      fd.close()
32
33  task_id = 0
34  post_data = {}
35  post_data["notification"] = "email"
36  post_data["email"] = "mailbox@example.com"
37  post_data["test_subject"] = { "content" : content, "filename" : file }
38  response = httputils.httprequest(SD['submit_url'], post_data)
39  if (response.status == 302 and
40      response.getheader("taskid", "DEFAULT") != "DEFAULT"):
41      task_id = response.getheader('taskid')
42      if not SD.has_key('insert'):
43          SD['insert'] = plpy.prepare("INSERT INTO reports (md5, provider
                , task_id, available) VALUES ($1, 'Anubis', $2, false)", [
                "text", "text" ])
44      plpy.execute(SD['insert'], [ md5, task_id ])
45      return 'true'
46  else:
47      plpy.notice(response.status)
48      plpy.notice(response.read())
49      return 'false'
50  $_$
51      LANGUAGE plpythonu;
```

B.6.11 submit_to_clamav

```
1  CREATE FUNCTION submit_to_clamav(md5 character) RETURNS boolean
2      AS $_$
3  import pyclamd
4  from pyclamd import ScanError
5  import os
6  from time import strftime, strptime
7
8  SAMPLES_PATH = '/var/www/uploads'
9  ALLAPLE_ID = 1
10 PROXIM_ID = 10
11
12 def prepare_query(name, query, args):
13     try:
14         if not SD.has_key(name):
15             SD[name] = plpy.prepare(query, args)
16             return
17     except NameError:
18         return
19
20 def execute_query(name, args):
21     try:
22         if SD.has_key(name):
23             return plpy.execute(SD[name], args)
24         else:
25             plpy.fatal("Query '%s' does not exist in SD, aborting." %(
                   name))
26     except NameError:
27          print "Running '%s' with args: %s" %(name, args)
28          return None
29
30 try:
31     pyclamd.ping()
32 except ScanError:
33     pyclamd.init_network_socket('localhost', 3310)
34
35 version = pyclamd.version().split('/')
36 scanner_version = version[0].split(' ')[1]
37 signature_version = strftime('%Y%m%d',strptime(version[2]))
38
39 prepare_query('getsample', 'SELECT sample FROM samples WHERE md5 = $1',
       [ 'text' ])
40 #prepare_query('checkvariant', 'SELECT * FROM clamav_variants WHERE
       name = $1 AND variant = $2', [ 'text', 'text' ])
41 prepare_query('newvariant', 'INSERT INTO clamav_variants(name, variant)
       VALUES ($1, $2)', [ 'text', 'text' ])
42 prepare_query('setsample', 'SELECT set_sample($1, $2, $3)', [ 'text','
       malware_type','int4'])
43 prepare_query('addreport', 'INSERT INTO av_reports (md5, scanner,
       scanner_version, signature_version, classification) VALUES ($1,
       $2, $3, $4, $5)', [ 'text', 'text', 'text', 'date', 'text'])
44
45 res = execute_query('getsample', [ md5 ])
46 if len(res) == 0:
47     plpy.error('Unknown sample %s' %(md5))
48 else:
49     sample = res[0]['sample']
50     file = "%s/%s" %(SAMPLES_PATH, md5)
51     try:
```

```
52          if sample != None:
53              result = pyclamd.scan_stream(sample)
54          elif os.path.exists(file):
55              result = pyclamd.scan_file(file)
56          else:
57              plpy.error('Cannot find sample file %s' %(md5))
58      except ScanError:
59          plpy.warning('Error scanning sample %s' %(md5))
60          return True
61
62      if result == None:
63          execute_query('addreport', [ md5, 'ClamAV-local',
                scanner_version, signature_version, 'OK' ])
64          return True
65      else:
66          virus = result.values()[0].strip()
67          execute_query('addreport', [ md5, 'ClamAV-local',
                scanner_version, signature_version, virus ])
68          lovirus = virus.lower()
69          if lovirus.find('allaple') != -1:
70              # process allaple
71              execute_query('setsample', [ md5, 'worm', ALLAPLE_ID ])
72              return False
73          elif virus == 'W32.Virut.A':
74              #res = execute_query('checkvariant', [ 'virut', virus ])
75              execute_query('setsample', [ md5, 'irc_bot', PROXIM_ID ])
76              return False
77          else:
78              return True
79              """
80  ··········if len(res) == 0:
81  ··········#new variant, register and process as usual
82  ··········execute_query('newvariant', ['virut', virus])
83  ··········return True
84  ··········else:
85  ··········#old variant, skip
86  ··········execute_query('setsample', [md5, 'irc_bot', ALLAPLE_ID
        ])
87  ··········return False
88  ··········"""
89  $_$
90      LANGUAGE plpythonu;
```

B.6.12 submit_to_cwsandbox

```
1  CREATE FUNCTION submit_to_cwsandbox(md5 character) RETURNS boolean
2      AS $_$
3  import sys
4  sys.path.append('/home/davide')
5  import httputils
6  import re
7  import urllib
8  import urlparse
9  from pg import unescape_bytea
10
11 def process_match(str, provider):
12     if not SD.has_key('link_r'):
```

```
13          SD['link_r'] = re.compile('.*<a␣href.*id=([0-9]+)\&amp\;
                password=([a-z]+).*', re.M)
14      link_m = SD['link_r'].search(str)
15      if link_m != None:
16          id = link_m.group(1)
17          passwd = link_m.group(2)
18          if not SD.has_key('insert'):
19              q = "INSERT␣INTO␣reports␣(md5,␣provider,␣analysis_id,␣
                    password,␣available)␣VALUES␣($1,␣$2,␣$3,␣$4,␣true)"
20              SD['insert'] = plpy.prepare(q, [ "text", "provider", "int4"
                    , "text"])
21          plpy.execute(SD['insert'], [ md5, provider, id, passwd ])
22          return True
23      else:
24          plpy.warning('failed␣link␣match')
25          return False
26
27  if not SD.has_key('selcheck'):
28      SD['selcheck'] = plpy.prepare(
29          "SELECT␣md5,␣provider␣FROM␣reports␣WHERE␣md5␣=␣$1␣and␣(provider
                ␣=␣'CWSandbox'␣or␣provider='VirusTotal')",
30          [ "text" ])
31
32  res = plpy.execute(SD['selcheck'], [ md5 ])
33  if len(res) == 1:
34      skip_provider = res[0]['provider']
35  elif len(res) > 1:
36      return 'false'
37  else:
38      skip_provider = None
39
40  if not SD.has_key('submit_url'):
41      res = plpy.execute("select␣value␣as␣submit_url␣from␣settings␣where␣
            key␣=␣'CWSANDBOX_SUBMIT_URL'")
42      SD['submit_url'] = res[0]['submit_url']
43
44  if not SD.has_key('sample_check'):
45      SD['sample_check'] = plpy.prepare('SELECT␣sample␣FROM␣samples␣WHERE
            ␣md5␣=␣$1␣AND␣sample␣IS␣NOT␣NULL', [ "text" ])
46
47  res = plpy.execute(SD['sample_check'], [ md5 ])
48  if len(res) > 0:
49      file = md5
50      content = unescape_bytea(res[0]['sample'])
51  else:
52      file = "/var/www/uploads/%s" %(md5)
53      fd = open(file, "rb")
54      content = fd.read()
55      fd.close()
56
57  post_data = {}
58  post_data['email'] = 'mailbox@example.com'
59  post_data["upfile"] = { "content" : content, "filename" : file }
60  response = httputils.httprequest(SD['submit_url'], post_data)
61  r = response.read()
62  if r.startswith('http://'):
63      url = r
64  else:
```

```
65      banner_r = re.compile('.*Please␣take␣a␣look␣at.*id=([0-9]+)&amp;
            password=([a-z]+).*',re.M|re.S)
66      m = banner_r.search(r)
67      if m != None:
68          url = 'http://www.cwsandbox.org/?page=samdet&id=' + m.group(1)
                + '&password=' + m.group(2)
69      elif r.find("Submission␣added.␣Thank␣you.␣Upon␣completion␣of␣the␣
            analysis␣you␣will␣find␣it␣on") != -1:
70          plpy.notice("new␣sample␣sumbitted␣for␣analysis")
71          return 'true'
72      elif r.find("You␣have␣already␣submitted␣this␣sample.␣Upon␣
            completion␣of␣the␣analysis␣you␣will␣find␣it␣on") != -1:
73          plpy.notice('old␣sample,␣analysis␣in␣progress')
74          return 'true'
75      else:
76          plpy.notice('unknown␣reply')
77          plpy.notice(r)
78          return 'true'
79  plpy.notice(url)
80  f = urllib.urlopen(url)
81  s = f.read()
82  f.close()
83  val = 'false'
84  idx = s.find('>Analysis</a>')
85  if idx != -1:
86      virustotal_r = re.compile('VirusTotal␣Scan␣(.*)Analysis</a>',re.S)
87      cwsandbox_r = re.compile('CWSandbox␣[0-9]+(.*)Analysis</a>',re.S)
88      plpy.notice("found␣page")
89      cws_m = cwsandbox_r.search(s)
90      if cws_m != None and skip_provider != 'CWSandbox':
91          plpy.notice("cwsandbox␣match")
92          str = cws_m.group(1)
93          if str.find('Error␣#1:␣The␣sample␣is␣no␣valid␣Win32␣application
                ') != -1:
94              plpy.notice("invalid␣sample")
95              query = plpy.prepare("UPDATE␣samples␣SET␣type='invalid'␣
                    WHERE␣md5␣=␣$1",['text'])
96              plpy.execute(query, [ md5 ])
97          elif str.find('Error␣#255:␣An␣unknown␣error␣happened␣during␣
                analysis') != -1 or str.find('Error␣#3:␣An␣error␣happened␣
                during␣analysis') != -1:
98              plpy.notice("analysis␣error")
99          elif str.find('analysis␣running') != -1:
100             plpy.notice("analysis␣running")
101         elif process_match(str, 'CWSandbox'):
102             val = 'true'
103     virus_m = virustotal_r.search(s)
104     if virus_m != None and skip_provider != 'VirusTotal':
105         plpy.notice("virustotal␣match")
106         if process_match(virus_m.group(1), 'VirusTotal'):
107             val = 'true'
108 return val
109 $_$
110     LANGUAGE plpythonu;
```

B.7 Trigger

B.7.1 do_submit

```
1  CREATE or REPLACE FUNCTION "public"."do_submit"()
2  RETURNS "pg_catalog"."trigger" AS
3  $BODY$
4  DECLARE
5          val boolean DEFAULT TRUE;
6  BEGIN
7        SELECT submit_to_clamav(NEW.md5) INTO val;
8        IF val IS TRUE THEN
9            PERFORM submit_to_anubis(NEW.md5);
10           PERFORM submit_to_cwsandbox(NEW.md5);
11       END IF;
12       RETURN NULL; -- ignored, this is run as AFTER trigger
13 END;
14 $BODY$
15 LANGUAGE 'plpgsql' VOLATILE;
```

B.7.2 set_geo

```
1  CREATE or REPLACE FUNCTION "public"."set_geo"()
2  RETURNS "pg_catalog"."trigger" AS
3  $BODY$
4  import GeoIP
5  if TD['event'] == 'INSERT':
6      if not SD.has_key('geoip'):
7          SD['geoip'] = GeoIP.open('GeoLiteCity.dat', GeoIP.
               GEOIP_MEMORY_CACHE)
8      data = SD['geoip'].record_by_addr(TD['new']['address'])
9      if data != None:
10         TD['new']['country'] = data['country_code']
11         TD['new']['latitude'] = data['latitude']
12         TD['new']['longitude'] = data['longitude']
13         return 'MODIFY'
14 return 'OK'
15 $BODY$
16 LANGUAGE 'plpythonu' VOLATILE;
```

Bibliografia

AA. VV. libvirt virtualization API, 2008. URL `http://libvirt.org/`.

Amini P. Kraken botnet infiltration, 2008. URL `http://dvlabs.tippingpoint.com/blog/2008/04/28/kraken-botnet-infiltration`.

Antonatos S., Anagnostakis K., e Markatos E. Honey@Home: a new approach to large-scale threat monitoring. In *WORM '07: Proceedings of the 2007 ACM workshop on Recurring malcode*, pages 38–45, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-886-2. doi: http://doi.acm.org/10.1145/1314389.1314398.

Baecher P., Koetter M., Holz T., Dornseif M., e Freiling F. The Nepenthes platform: An efficient approach to collect malware. In Springer, (A cura di), *Proceedings of the 9th International Symposium On Recent Advances In Intrusion Detection (RAID)*, pages 165–184, settembre 2006. URL `http://honeyblog.org/junkyard/paper/collecting-malware-final.pdf`.

Balas E. e Viecco C. Towards a third generation data capture architecture for honeynets. In *Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 2005. Proceedings from the Sixth Annual IEEE*, pages 21–28, giugno 15–17, 2005. doi: 10.1109/IAW.2005.1495929. URL `http://www.honeynet.org/papers/individual/hflow.pdf`.

Barford P. e Yegneswaran V. An inside look at botnets. In Springer, (A cura di), *Malware Detection*, volume 27 of *Advances in Information Security*, 2006. URL `http://pages.cs.wisc.edu/~vinod/botnets.pdf`.

Bayer U., Moser A., Kruegel C., e Kirda E. Dynamic analysis of malicious code. *Journal in Computer Virology*, 2(1):67–77, agosto 2006. ISSN 1772-9890. doi: 10.1007/s11416-006-0012-2. URL `http://dx.doi.org/10.1007/s11416-006-0012-2`.

Bellard F. QEMU, a fast and portable dynamic translator. In *Proceedings of the USENIX 2005 Annual Technical Conference, FREENIX Track*, pages 41–46, giugno 2005. URL `http://www.usenix.org/publications/library/proceedings/usenix05/tech/freenix/full_papers/bellard/bellard.pdf`.

Black J., Cochran M., e Highland T. *A Study of the MD5 Attacks: Insights and Improvements*, volume 4047/2006 of *Lecture Notes in Computer Science*, chapter Fast Software Encryption, pages 262–277. Springer Berlin / Heidelberg, ottobre 2006. doi: 10.1007/11799313. URL `http://www.cs.colorado.edu/~jrblack/papers/md5e-full.pdf`.

Dornseif M., Holz T., e Klein C. N. NoSEBrEaK - attacking honeynets. In *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, pages 123–129, giugno 2004. doi: 10.1109/IAW.2004.1437807. URL `http://md.hudora.de/publications/2004-NoSEBrEaK.pdf`.

Eichin M. W. e Rochlis J. A. A. With microscope and tweezers: An analysis of the internet virus of november 1988. In *Proceedings of the 1989 IEEE Computer Society Symposium on Security and Privacy*, Oakland, Ohio, 1989. URL `http://www.mit.edu/~eichin/virus/main.html`.

F-Secure. Malware information pages: Allaple.A, dicembre 2006. URL `http://www.f-secure.com/v-descs/allaple_a.shtml`.

F-Secure. F-Secure virus descriptions: Melissa, 2001a. URL `http://www.f-secure.com/v-descs/melissa.shtml`.

F-Secure. F-Secure virus descriptions: Nimda, settembre 2001b. URL `http://www.f-secure.com/v-descs/nimda.shtml`.

Franklin J., Paxson V., Perrig A., e Savage S. An inquiry into the nature and causes of the wealth of internet miscreants. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 375–388, New York, NY, USA, 2007. ACM. doi: 10.1145/1315245.1315292. URL `http://www.cs.cmu.edu/~jfrankli/acmccs07/ccs07_franklin_eCrime.pdf`.

Göbel J. G. Amun: Python honeypot, 2008a. URL `http://amunhoney.sourceforge.net`.

Göbel J. G. Malware extracts CD keys, giugno 2008b. URL `http://zeroq.kulando.de/post/2008/06/09/malware-extracts-cd-keys`.

Göbel J. G. Infiltrator v0.1, novembre 2007. URL `http://zeroq.kulando.de/post/2007/11/15/infiltrator_v01`.

Google. Google Maps Italia, 2008. URL `http://maps.google.it`.

Govil J. e Govil J. Criminology of botnets and their detection and defense methods. In *Electro/Information Technology, 2007 IEEE International Conference on*, pages 215–220, Chicago, IL, USA, maggio 2007. doi: 10.1109/EIT.2007.4374517.

Granger S. Social engineering fundamentals, dicembre 2001. URL `http://www.securityfocus.com/infocus/1527`.

Grizzard J. B., Sharma V., Nunnery C., Kang B. B., e Dagon D. Peer-to-peer botnets: overview and case study. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 1–1, Berkeley, CA, USA, 2007. USENIX Association. URL `http://www.usenix.org/event/hotbots07/tech/full_papers/grizzard/grizzard_html/`.

Hispasec Sistemas. VirusTotal, 2008. URL `http://www.virustotal.com`.

Holz T. e Raynal F. Detecting honeypots and other suspicious environments. In *Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC*, pages 29–36, giugno 2005. doi: 10.1109/IAW.2005.1495930. URL `http://www.honeynet.org/papers/individual/DefeatingHPs-IAW05.pdf`.

Holz T. CWSandbox vs. ALLAPLE, aprile 2007a. URL `http://honeyblog.org/archives/110-CWSandbox-vs.-ALLAPLE.html`.

Holz T. Security of virtual machines, aprile 2007b. URL `http://honeyblog.org/archives/109-Security-of-virtual-machines.html`.

Holz T., Steiner M., Dahl F., Biersack E., e Freiling F. Measurements and mitigation of peer-to-peer-based botnets: A case study on Storm worm. In *1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '08), San Francisco, CA, USA*, aprile 2008. URL `http://honeyblog.org/junkyard/paper/storm-leet08.pdf`.

International Secure Systems Lab. Anubis: Analyzing unknown binaries, 2008. URL `http://anubis.iseclab.org`.

Jiang X., Xu D., e Wang Y.-M. Collapsar: a VM-based honeyfarm and reverse honeyfarm architecture for network attack capture and detention. *Journal of Parallel and Distributed Computing*, 66(9):1165–1180, settembre 2006. doi: 10.1016/j.jpdc.2006.04.012. URL `http://www.cs.gmu.edu/~xjiang/pubs/JPDC06.pdf`.

Joe Security. Joebox, 2008. URL `http://www.joebox.org/`.

Kaspersky Lab. Virus.Win32.Gpcode.ak, giugno 2008. URL `http://www.viruslist.com/en/viruses/encyclopedia?virusid=313444`.

Kiszka P. Gmapper — Google Maps in PHP, 2008. URL `http://gmapper.ajax-info.de`.

Lyon G. Nmap, 2008. URL `http://nmap.org`.

Metasploit LLC. The Metasploit project, 2008. URL `http://www.metasploit.com/`.

Moore D. e Shannon C. The spread of the code-red worm (crv2). Relazione tecnica, CAIDA, 2001. URL `http://www.caida.org/research/security/code-red/coderedv2_analysis.xml`.

Nazario J. BlackEnergy DDoS bot analysis. Relazione tecnica, Arbor Networks, ottobre 2007. URL `http://asert.arbornetworks.com/2007/10/blackenergy-ddos-bot-analysis-available/`.

Norman. SandBox information center, 2008. URL `http://sandbox.norman.no/`.

Oberheide J. Detecting and evading CWSandbox, gennaio 2008. URL `http://jon.oberheide.org/blog/2008/01/15/detecting-and-evading-cwsandbox/`.

Ollmann G. The phishing guide. Relazione tecnica, Technical Info, 2006. URL `http://www.technicalinfo.net/papers/Phishing.html`.

Ormandy T. An empirical study into the security exposure to hosts of hostile virtualized environments. Relazione tecnica, Google, Inc., aprile 2007. URL http://taviso.decsystem.org/virtsec.pdf.

Portokalidis G., Slowinska A., e Bos H. Argos: an emulator for fingerprinting zero-day attacks. In *Proceedings of the first ACM SIGOPS EUROSYS'2006*, volume 40, pages 15–27, Leuven, Belgium, aprile 2006. ACM. URL http://www.cs.vu.nl/~herbertb/papers/argos_eurosys06.pdf.

Pouget F., Dacier M., e Pham V. H. Leurre.com: on the advantages of deploying a large scale distributed honeypot platform. In *ECCE'05, E-Crime and Computer Conference, 29-30th March 2005, Monaco*, marzo 2005. URL http://www.leurrecom.org/papers.php.

Provos N. A virtual honeypot framework. In *Proceedings in the 13th USENIX Security Symposium*, San Diego, CA, agosto 2004. URL http://www.citi.umich.edu/u/provos/papers/honeyd.pdf.

Rajab M. A., Zarfoss J., Monrose F., e Terzis A. A multifaceted approach to understanding the botnet phenomenon. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 41–52, New York, NY, USA, 2006. ACM. ISBN 1-59593-561-4. doi: http://doi.acm.org/10.1145/1177080.1177086. URL http://www.imconf.net/imc-2006/papers/p4-rajab.pdf.

Seifert C., Steenson R., Holz T., Bing Y., e Davis M. A. Know your enemy: Malicious web servers. Relazione tecnica, The Honeynet project, 2007a. URL http://www.honeynet.org/papers/mws/.

Seifert C., Welch I., e Komisarczuk P. HoneyC: The low-interaction client honeypot. In *Proceedings of the 2007 NZCSRCS.*, Hamilton, New Zealand, aprile 2007b. Waikato University. URL http://www.mcs.vuw.ac.nz/~cseifert/blog/images/seifert-honeyc.pdf.

Sophos. W32/Allaple-B, 2008. URL http://www.sophos.com/security/analyses/viruses-and-spyware/w32allapleb.html.

Sourcefire, Inc. Clam antiVirus, 2008. URL http://www.clamav.net.

SUN Microsystems. VirtualBox, 2008. URL http://www.virtualbox.org/.

Sunbelt. CWSandbox, 2008. URL http://www.cwsandbox.org/.

Tenable Network Security. Nessus, 2008. URL http://www.nessus.org/nessus/.

The Artemis Team. HoneyBow, 2008. URL http://honeybow.mwcollect.org/.

The Honeynet project. Capture-HPC client honeypot / honeyclient, 2008a. URL https://projects.honeynet.org/capture-hpc.

The Honeynet project. Know your enemy: Fast-flux service networks, giugno 2007. URL http://www.honeynet.org/papers/ff/fast-flux.html.

The Honeynet project. HoneyC, 2008b. URL `https://projects.honeynet.org/honeyc`.

The Honeynet project. Know your enemy: Honeywall cdrom roo, agosto 2005. URL `http://www.honeynet.org/papers/cdrom/roo/index.html`.

The Linux Foundation. Net:bridge, 2008. URL `http://www.linux-foundation.org/en/Net:Bridge`.

The mwcollect alliance, 2008. URL `http://alliance.mwcollect.org`.

The NoAH project, 2008. URL `http://www.fp6-noah.org`.

Trend Micro. 2007 threat report and forecast. Relazione tecnica, Trend Micro, 2007. URL `http://trendmicro.mediaroom.com/index.php?s=65&item=163`.

Virgilio. Aeneid. In Mynors R. A. B., (A cura di), *P. Vergili Maronis opera*. Oxford, 1969.

VMWare, Inc. VMWare, 2008. URL `http://www.vmware.com`.

Werner T. Honeytrap, 2008. URL `http://honeytrap.mwcollect.org/`.

Willems C., Holz T., e Freiling F. Toward automated dynamic malware analysis using CWSandbox. *IEEE Security & Privacy Magazine*, 5(2):32–39, marzo/aprile 2007. ISSN 1540-7993. doi: 10.1109/MSP.2007.45.

XenSource. Xen, 2008. URL `http://www.xen.org`.

Zhuge J., Holz T., Han X., Song C., e Zou W. Collecting autonomous spreading malware using high-interaction honeypots. In *ICICS 2007*, pages 438–451, 2007. doi: 10.1007/978-3-540-77048-0_34. URL `http://honeyblog.org/junkyard/paper/honeybow-ICICS07.pdf`.

Zou C. C. e Cunningham R. Honeypot-aware Advanced Botnet Construction and Maintenance. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 199–208, Philadelphia, PA, 2006. doi: 10.1109/DSN.2006.38. URL `http://www.cs.ucf.edu/~czou/research/honeypot-DSN06.pdf`.

Indice analitico