Scuola Universitaria Professionale
della Svizzera Italiana

SUPSI

Dipartimento
Tecnologie
Innovative

Lavoro di diploma
I-03/04-D-30

Virus scanner for open source firewall

Studente: Luca Comparato

Docente responsabile: Angelo Consoli, Federico Flückiger

Committente: Angelo Consoli

Data: 15 ottobre 2004

Sommario

Questo lavoro di diploma permette di approfondire le nozioni sulla sicurezza dei sistemi e delle reti, ricevute durante il corso di telematica. Lo scopo del presente lavoro è l'estensione delle funzionalità del sistema firewall *m0n0wall* aggiungendovi un sistema di antivirus. Si tratta pertanto di un lavoro di sviluppo software ed integrazione in un architettura già esistente

Abstract

This diploma job allows me to deepen my knowledge in network and system security acquired during the data transmission course. The project aim is to extend *mOnOwall* firewall's functions by adding an antivirus system. Therefore this is a software development and integration within an already existing architecture.

Ringraziamenti

Ringrazio mia mamma per avermi permesso di studiare ed imparare questa affascinante professione. Ringrazio la mia fidanzata per essermi stata vicina anche nei momenti di sconforto.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Definizione del progetto

1.1 Progetto assegnato:

Ciclo di studio: Informatica

Catalogazione: I-03/04-D-30

Titolo del progetto: Virus scanner for open source firewall

Considerazioni: Il presente lavoro permette di approfondire le nozioni sulla sicurezza dei sistemi e delle reti ricevute durante il corso di telematica. Da alcuni anni alla SUPSI vengono analizzati diversi firewall, tra i quali pure alcuni basati su software open source, e nel corso del tempo di è cominciato ad approfondire il tema per realizzare autonomamente un apparecchio di questo tipo.

Obiettivi: Lo scopo del presente lavoro è l'estensione delle funzionalità del sistema attualmente esistente in fase di sviluppo fornito dal docente. Si tratta pertanto di un lavoro di sviluppo software ed integrazione in un architettura già esistente.

Compiti: Analizzare il software attualmente in uso e definire una strategia di scansione del traffico da virus. Ricercare su web alcuni pacchetti antivirus open source. Studiare i software reperiti e scegliere il più idoneo all'implementazione, prestando particolare attenzione ai protocolli con i quali questi possono operare. Studiare e realizzare un sistema di logging efficiente e conseguentemente un sistema di analisi dei log che lo sfrutti. Realizzare l'interfaccia web, sia per la configurazione del sistema di scansione che per la visualizzazione dei log di dati, da integrare nell'applicativo esistente (programmazione in PHP su base MySQL). Studiare e realizzare una pagina riassuntiva dello stato del sistema (statistiche, uptime delle connessioni protette e non, carico della CPU, utilizzo della memoria, larghezza di banda utilizzata sulle varie interfacce, ...). Buona documentazione del lavoro

Docente responsabile: Angelo Consoli, Federico Flückiger

Committente: Angelo Consoli

1.2 Piano dei lavori:

PREVISTO

Settimana numero	1	2	3	4	5	6
	6->12.09	13->19.09	20->26.09	27->03.10	04->10.10	11->14.10
Ricerca						
Ricerca antivirus open source						
Installazione rete x simulazione						
Studio di PHP						
Sudio firewall m0n0						
Sviluppo						
Sviluppo pagine settaggi AV						
Sviluppo pagine riassuntive						
Porting su m0n0						
Test						
Documentazione						

Figura 1.1: Piano lavori previsto.

EFFETTIVO

Settimana numero	1	2	3	4	5	6
	6->12.09	13->19.09	20->26.09	27->03.10	04->10.10	11->14.10
Ricerca						
Ricerca antivirus open source						
Installazione rete x simulazione						
Studio di PHP						
Sudio firewall m0n0						
Ricerca sistema di web caching						
Sviluppo						
Sviluppo pagine settaggi AV						
Sviluppo pagine riassuntive						
Integrazione Squid + Squivi + AV						
Risoluzione problemi						
Porting su m0n0						
Test						
Documentazione						

Figura 1.2: Piano lavori effettivo.

Capitolo 2

Introduzione

2.1 Introduzione:

Il firewall m0n0wall[1] è un sistema firewall che può essere utilizzato sia su un PC che su un'architettura più semplice. Comprende tutte le funzionalità di un prodotto commerciale con il pregio di essere gratuito. È basato su una versione minimale di FreeBSD, comprendente un web server, PHP e altre *utility*. L'intera configurazione del firewall infine, è contenuta in un file XML.
Il firewall m0n0 è probabilmente il primo sistema UNIX ad avere una fase di boot ultimata con il linguaggio PHP, in luogo dei canonici script di shell.

Figura 2.1: Firewall m0n0wall.

L'autore di m0n0 si chiama Manuel Kasper ed ha presentato una prima beta release del firewall nel febbraio del 2003. La versione 1.0 dello stesso è stata rilasciata esattamente un anno più tardi.

[1] http:/www.m0n0.ch/wall

Le principali caratteristiche di questo prodotto sono le seguenti:

- Interfaccia web con supporto SSL[2]
- Console con interfaccia per il *recovery*
  - Per settare l'IP della LAN
  - Per settare una password
  - Impostare i settaggi di *default*
  - Eseguire il reboot del sistema
- Supporto per il wireless
- Packet filtering
  - regole di passaggio/blocco
  - logging
- NAT/PAT (incluso 1:1)
- Supporto DHCP client, PPPoE e PPTP per interfacce WAN
- IPsec VPN tunnels
- PPTP VPN
- Static routes
- DHCP server
- Caching DNS forwarder
- DynDNS client
- SNMP agent
- traffic shaper
- Possibilità di aggiornamento del firmware dall'interfaccia web

2.2 Requisiti:

M0n0wall è stato sviluppato per operare su PC embedded su base X86. I sistemi net45xx/net48xx dell'azienda Soekris Engineering[3] e le piattaforme WRAP dell'azienda PC Engines[4] sono supportati in modo nativo. Con questi sistemi si può tranquillamente scaricare l'immagine di m0n0 e scriverla su di una CF card (da 8MB o superiori), e non resta che configurare il sistema.
È possibile naturalmente eseguire m0n0wall sulla maggior parte dei normali PC, utilizzando un Harddisk, una CF card oppure la versione per CD-ROM che necessita di un floppy per il salvataggio delle impostazioni.
Il quantitativo di memoria RAM per poter eseguire m0n0wall è fissato a 64MB o superiori.

[2]Secure Sockets Layer
[3]http://www,soekris.com
[4]http://www.pcengines.ch

Capitolo 3

Componenti principali

Per sviluppare al meglio il modulo Antivirus per il firewall m0n0wall, sono richieste le seguenti componenti:

- un sistema per lo sviluppo: un computer con installato il sistema operativo FreeBSD versione 4.9 con permessi di amministratore (root)
- un sistema per testare il Firewall: un sistema embedded oppure un PC sul quale sia installato il firewall m0n0wall
- un sistema che simuli una rete di computer: un paio di PC con eventualmente uno switch per potersi assicurare che tutto funzioni nella maniera ottimale
- Alcuni scripts: per il firewall m0n0wall è disponibile almeno uno script scritto in PHP, questo permette di comprimere e decomprimere l'immagine del sistema firewall per le necessarie modifiche e/o aggiunte, ma è possibile svolgere queste operazioni da linea di comando.

Capitolo 4

Virus informatici

4.1 Che cosa è un virus informatico:

Un virus informatico è un programma, cioè una serie di istruzioni scritte da un programmatore ed eseguibili da un computer, che ha le seguenti caratteristiche: è stato scritto per inglobarsi e confondersi alle istruzioni di altri programmi modificandoli; chi lo ha scritto ha previsto la possibilità che il virus sia in grado di replicarsi, ovvero di copiare le istruzioni che lo compongono in altri programmi. Dopo un tempo prestabilito, necessario per effettuare la replicazione, il virus comincia a compiere l'azione per cui è stato scritto, che può consistere, per esempio, nel distruggere dati e/o programmi presenti su di un supporto magnetico o, semplicemente, nel far comparire a video un messaggio.

Da ciò si deduce che: i virus non sono capaci di un comportamento autonomo: tutto ciò che sono in grado di fare è stato puntualmente previsto - come per qualsiasi programma - dai programmatori che li hanno ideati e scritti; i virus sono facilmente identificabili ed eliminabili da programmi - detti antivirus - scritti appositamente. Questi ricercano negli altri programmi presenti sul computer la sequenza di istruzioni che caratterizza il virus; ciò è però possibile solo se i virus sono noti, e cioè se è nota, almeno in parte, la sequenza di istruzioni con cui sono stati scritti: tale sequenza è diversa per ogni virus.

I virus, come tutti i programmi, non possono funzionare, e quindi portare a termine il compito loro assegnato, se non nel sistema per cui sono stati scritti; quindi un virus scritto per computer che usano il sistema operativo Windows, non potrà funzionare su computer Macintosh.
I virus informatici (virus in latino significa veleno) hanno mutuato il loro nome dal campo medico - biologico, per una vaga somiglianza con alcune caratteristiche dei virus nella microbiologia: come questi ultimi, per riprodursi, devono penetrare in una cellula ospite ed assumere il controllo dei suoi processi metabolici.
Così i virus informatici devono penetrare nel programma ospite modificandolo, sia per riprodursi sia, in seguito, per danneggiare dati e/o programmi presenti su supporti registrabili. Come nella biologia i virus sono organismi relativamente semplici e molto piccoli, rispetto all'organismo che invadono, così anche i virus informatici sono dei programmi costituiti da poche centinaia di istruzioni, al massimo un migliaio; ciò consente loro di portare a termine il compito per cui sono stati scritti senza, in genere, far notare la loro presenza all'utente del computer.
Il primo virus fu sviluppato nel novembre del 1983, con fini dimostrativi, nell'ambito di

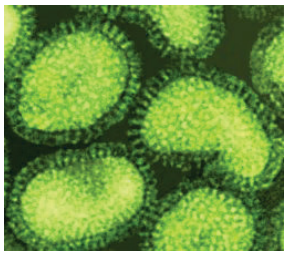

Figura 4.1: Virus dell'influenza rilevato al microscopio.

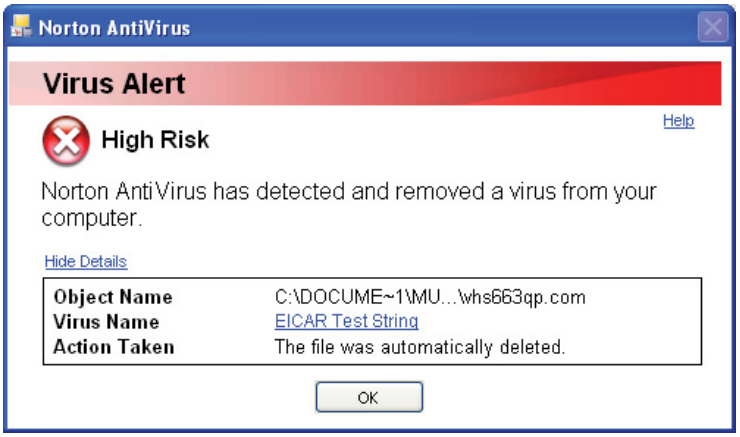

Figura 4.2: Virus informatico rilevato da un antivirus commerciale.

una ricerca finanziata da una delle principali società costruttrici di computer ed inserita in un più generale progetto di studio della sicurezza dei sistemi informativi. L'obiettivo della ricerca era di dimostrare come le possibilità di un attacco al patrimonio informativo di un'azienda non fossero limitate a quelle tradizionalmente prese in esame negli studi sulla sicurezza fino ad allora svolti. Tali studi infatti avevano incentrato la loro attenzione sull'attacco fisico (p.es. atti terroristici), sulla conoscenza illegittima di password e su modifiche ai programmi effettuate da personale interno alle aziende.

L'esperimento, che riuscì perfettamente, dimostrò che predisponendo opportunamente il programma aggressore era possibile attaccare qualsiasi sistema: il virus sperimentale, sviluppato in sole otto ore da un esperto, impiegava meno di mezzo secondo per replicarsi copiandosi in un altro programma, che diventava, a sua volta, portatore del virus.

4.2 Classificazione dei virus informatici:

Di seguito una breve spiegazione dei vari tipi di virus a seconda della loro classificazione:

Virus: Sono dei software che colpiscono i file di altri programmi modificandoli inserendo al loro interno il proprio codice (che così può riprodursi), causando così svariati tipi di danno (molto diversi tra loro per tecnica utilizzata nel crearli e per gravità).Molto frequentemente i virus colpiscono il file sorgente dei programmi da infettare, poiché in questo modo possono replicarsi ad ogni avvio del programma.

I virus tendono sempre a replicarsi (cioè a creare copie a se identiche) che possano diffondersi a loro volta, spargendo quindi il loro codice di file in file, di computer in computer.

Trojan Horse: Si tratta di una categoria a parte di programmi, non atta a creare veri e propri danni al pc ma bensì a rendere il sistema più vulnerabile ad altre forme di attacco.

I trojan horse possono entrare nel PC attraverso la posta elettronica, o magari scaricandoli da siti poco attendibili. Essi infatti sono quasi sempre mascherati da patch per altri programmi che dovrebbero migliorarne le prestazioni.

In realtà non fanno altro che (a seconda del tipo) inviare informazioni riservate dell' utente al creatore del virus oppure, in casi più tragici, permettere ad un utente remoto (quasi sempre con pessime intenzioni) di prendere il controllo del pc, causando ogni sorta di danni.

Worm: È un piccolo programma che sfrutta le reti per moltiplicarsi. Hanno la capacità di diffondere il proprio codice funzionante (o parte di esso) in altri sistemi, per replicarsi ancora ed ancora fino alla completa saturazione della rete.

Il worm mira proprio a questo: vuole replicarsi così tanto e così in fretta da intasare le comunicazioni informatiche ed intasare i server. In passato alcuni worm ci sono riusciti molto bene come ad esempio il worm *Melissa*.

A differenza dei virus non hanno bisogno di un programma che li ospiti, poichè in genere si autoallegano ai messaggi di posta che il computer infetto invia.

Backdoor: La backdoor è l'ultimo esempio di virus. Una backdoor, permette al nostro computer di diventare un server della rete quindi di far prendere possesso della nostra macchina chiunque.

Le backdoor più diffuse sono BO (Back Orifice, Il nome è tutto un programma) e NetBus. Esse sfruttano i malfunzionamenti costruttivi di Windows, permettendo l'accesso al PC tramite le ormai famose porte UDP che questo sistema operativo lascia aperto.

Capitolo 5

Antivirus ClamAV

5.1 Introduzione:

Il software antivirus utilizzato **ClamAV**, è la versione riscritta in linguaggio C di OpenAntivirus, un primo timido tentativo di antivirus GPL sviluppato in Java che non aveva conosciuto un grande successo e presentava grossi problemi di scalabilità.

Figura 5.1: ClamAV.

L'autore di ClamAV pose l'accento sul fatto che l'antivirus avrebbe garantito ottime performance e che il database con le signature dei virus sarebbe stato mantenuto aggiornato. Questo ha permesso a ClamAV di competere con i software commerciali che da sempre dominano il mercato. Per concludere, ClamAV non è prerogativa del mondo Unix, ma esiste anche una versione per Windows[1], anch'essa gratuita ed aggiornabile.

5.2 L'importanza degli update:

Istintivamente, viene da chiedersi perché non sia stato scritto un antivirus in grado di utilizzare il database creato per altri software, in modo da liberarsi della fatica di mantenerlo aggiornato con tempestività.

Ci sono due ragioni: la prima è che il database è coperto da copyright e può essere utilizzato solo dai titolari di una licenza dell'antivirus, la seconda è che tutte le licenze

[1] http://www.clamwin.net

prevedono una clausula che vieta il reverse engineering di qualsiasi parte del software. Addirittura alcune licenze vietano la condivisione dei risultati degli scan con terzi.
Tomasz Kojm, l'autore di ClamAV, decise così di proseguire sulla strada aperta da OpenAntivirus e creare un database di signature da distribuire con licenza GPL, cosicchè ci si liberasse una volta per tutte dell'oligopolio delle case produttrici di antivirus. L'importanza di questa scelta non deve essere sottovalutata: grazie a ClamAV, oggi, chiunque può scrivere il proprio motore per antivirus e riutilizzare il database esistente, senza violare alcuna legge sul copyright.

Il ClamAV Virus Database (cvd) comprende due file: uno di dimensioni ridotte (daily.cvd) aggiornato frequentemente ed uno più grande (main.cvd) che viene aggiornato una volta al mese, al fine di minimizzare il consumo di banda. Tale formato viene ampiamente descritto nella documentazione di ClamAV, cosicché chiunque può scrivere un applicazione in grado di riutilizzare il database.

5.3 La creazione delle signature:

L'impresa di costruire un database di virus pattern partendo da zero o quasi, si è rivelata decisamente mastodontica. Il primo problema è stato quello di ottenere i virus stessi o perlomeno copie di file infetti. All'inizio ci si è basati esclusivamente sui file sospetti rilevati da altri antivirus, la cui licenza non fosse eccessivamente restrittiva. In seguito si è deciso di avvalersi dell'aiuto della comunità di utenti ClamAV. Pian piano si è costituito un gruppo di persone dedite esclusivamente alla manutenzione ed all'aggiornamento del database, i membri di questo team sono stati subito simpaticamente ribattezzati *sigmakers*[2] che è la contrazione di signature markers ovvero creatori di firme.

La creazione delle signature è un'attività molto delicata. Il pattern che occorre trovare deve essere in grado di riconoscere anche esemplari leggermente modificati del virus, ma deve evitare falsi positivi (innoqui scambiati per infetti): per ottenere questo risultato si procede per approssimazioni successive effettuando di volta in volta lo scan di parecchi GB di dati che si sa con certezza essere privi di virus.

5.4 La distribuzione degli update:

Avere a disposizione un database sempre aggiornato è utile soprattutto nella misura in cui si riesce a distribuirlo velocemente a centinaia di migliaia di utenti. È cruciale per l'efficacia dell'antivirus stesso, che il worm sia riconosciuto appena cominci la sua diffusione: qualche decina di minuti di ritardo sono fatali. Attualmente il database viene distribuito tramite una serie di server che fanno da mirror (circa 50) che devono sopportare un traffico che si avvicina ai 2 TB/mese

5.5 Lo scan engine di ClamAV:

Lo scan engine è la parte dell'antivirus che effettua il pattern matching vero e proprio, e verifica che all'interno del file da analizzare non siano presenti le signature specificate nel

[2] http://www.clamav.net/team.html

database. Lo scanner è in grado di effettuare questa ricerca all'interno dei file in formato *raw*, così come vengono letti dal filesystem, ma può anche effettuare delle operazioni di decodifica e decompressione su di essi ed eseguire il pattern matching sui dati ottenuti. In particolare supporta la maggior parte degli archivi compressi, i file codificati in MIME o unencode ed inseriti all'interno di altri file di testo, come succede ad esempio con i messaggi di posta in formato mbox e Maildir e perfino le macro salvate all'interno di documenti di Microsoft Office. L'accesso agli archivi gzip, bzip2, tar, zip e rar 2.0 avviene direttamente tramite alcune funzioni della libreria *LibClamAV*, inoltre *clamscan*[3] può chiamare con una *system()* i decompressori esterni per i formati zip (UnZip, UnRar, UnAce,...) utilizzando appositi flag che permettono di usare il decompressore interno in favore di quello specificato da linea di comando.
Per prevenire attacchi di tipo **DoS** vengono effettuati dei controlli sul rapporto di compressione degli archivi, e sul numero di subdirectory e di file presenti al loro interno. Immaginatevi cosa succederebbe se il vostro mail server tentasse di analizzare il contenuto di un file di testo contente il solo carattere **A** ripetuto per qualche miliardo di volte e che una volta compresso occupa solamente pochi byte, oppure immaginate un archivio contenente qualche centinaio di sottodirectory: ben presto le risorse di sistema verrebbero esaurite.

5.6 Clamscan vs clamd e clamdscan:

Lo scan engine si trova all'interno della libreria LibClamAV e può essere chiamato in tre modi diversi: tramite *clamscan*, tramite il demone *clamd* ed il client *clamdscan*, oppure collegando il proprio programma direttamente alla libreria *LibClamAV* ed utilizzando le opportune funzioni come ampiamente descritto nella documentazione ufficiale. I primi due approcci sono comunque i più diffusi per quanto riguarda i mail server.
Esistono sostanziali differenze in termini di prestazioni e funzionalità tra *clamscan* e clamd+clamdscan. Ogni volta che viene avviato, *clamscan* carica in memoria il database con le signature ed esegue uno scan dei path passati come argomento. L'overhead causato dal caricamento in memoria del database ad ogni chiamata è considerevole, quindi l'utilizzo di *clamscan* in un ambiente come un proxy server, in cui occorre eseguire un scan su ogni singolo messaggio è inaccettabile. Il demone *clamd* permette di caricare in memoria il database solo all'avvio e di effettuare un reload ogni volta che il database viene aggiornato. Un'altra differenza consiste nell'impossibilità per *clamd* di chiamare dei decompressori esterni per accedere ad archivi che non siano supportati da *LibClamAV*. Il demone *clamd* si appoggia al file di configurazione *clamd.conf*[4], dove quasi ogni riga di comando per *clamscan* ha un corrispettivo nel file di configurazione, in tabella 5.1 vengono elencate le principali.
Una fattore da prendere in considerazione quando si utilizza *clamdscan* in luogo di *clamscan*, è che la scansione viene sempre eseguita con i privilegi dell'utente *clamd*, quindi come l'utente definito dalla direttiva *User* in *clamd.conf*, piuttosto che quelli dell'utente che esegue clamdscan. Per questo motivo non bisogna stupirsi se si ottiene un messaggio d'errore, dovuto a permessi di accesso non sufficienti.

[3] vedi Capitolo 5.6
[4] vedi Appendice **??** per la versione integrale

Linea di comando	File di configurazione	Descrizione
–max-recursion	ArchiveMaxRecursion	Massimo n° di subdirectory
–max-space	ArchiveMaxFileSize	Massimo spazio disponibile per la decompressione
–block-encrypted	ArchiveBlockEncrypted	Blocca i file criptati
–database	DatabaseDirectory	Directory contenente le signatures
–mbox	ScanMail	Tratta il file come una mailbox

Tabella 5.1: Tabella principali configurazioni clamd.conf.

5.7 Utilizzo di freshclam:

La maggior parte degli antivirus per il mondo *nix[5], non prevede un software apposito che si occupi dell'aggiornamento del database. Di solito si demanda questo compito ad uno script in linguaggio *bash* oppure *Perl* che viene lanciato da *crontab* ad intervalli regolari. L'autore di ClamAV ha preferito spingersi oltre ed ha realizzato *freshclam*. Si tratta di un programm in grado di funzionare in due modalità: se viene lanciato con il flag -d in entrata, entra in background ed aggiorna periodicamente il database, altrimenti esegue un aggiornamento ed esce immediatamente. *freshclam*, ad ogni iterazione, controlla tramite il comando *HEAD* del protocollo HTTP, se è disponibile una versione dei database *main.cvd* e *daily.cvd* più recente rispetto alle copie in locale: in tal caso il database viene scaricato per intero e viene verificata l'integrità del file andando a leggerne il checksum MD5 e la firma digitale.

L'utilizzo della firma digitale è un prerogativa di ClamAV assente nella maggior parte degli altri antivirus. Essa consente, con una certa sicurezza, di proteggersi da eventuali Denial of Service: pensate a cosa accadrebbe se un hacker riuscisse ad inserire una signature abbastanza generica dentro il database dell'antivirus: tutto verrebbe cestinato, perché identificato come virus.

Per difetto, *freshclam* effettua 36 iterazioni al giorno, il numero di controlli giornaliero può essere però modificato con la direttiva *Checks* nel suo file di configurazione *freshcalm.conf*. Nel caso si stia utilizzando *clamd*, occorre notificare al demone quando un nuovo database diviene disponibile, affinché possa caricare in memoria le nuove definizioni. *freshclam* può inviare a *clamd* il comando RELOAD quando nel suo file di configurazione viene aggiunta l'opzione *NotifyClamd*.

Infine nel file di configurazione è possibile definire l'hostname di uno o più mirror (DatabaseMirror), il numero massimo di tentativi da effettuare se la connessione fallisce la prima volta (MaxAttempts), un eventuale proxy HTTP da utilizzare per la connessione al mirror (HTTPProxyServer e HTTPProxyPort), le eventuali informazioni di login (HTTPProxyUsername e HTTPProxyPassword) ed eventuali comandi da lanciare ogni volta che un aggiornamento ha successo (OnUpdateExecute) o fallisce (OnErrorExecute). Tutte queste opzioni possono anche essere specificate direttamente da linea di comando.

[5]derivati di Unix

Capitolo 6

Transparent proxy: Squid

6.1 Introduzione:

Il concetto di **inline cache** combina in un unico apparato sia il webcaching che le tecniche di routing e/o bridging, questo apparato può essere configurato con due o più interfacce di rete. Il classico esempio di **inline cache** è una macchina UNIX sulla quale viene eseguito Squid, questo tipo di device viene posizionato sul percorso di rete per consentirgli di catturare il traffico HTTP e ridirezionarlo verso Squid. Non sono molti gli apparati dedicati disegnati per svolgere questo compito perchè una inline cache può rappresentare un single point of failure. Molti produttori infatti raccomandano di utilizzare sistemi sviluppati da terze parti come gli switch di livello quattro che garantiscono un maggior livello di disponibilità del servizio. Possiamo progettare una inline cache a basso costo utilizzando dei semplici PC che vengono equipaggiati con GNU Linux o FreeBSD e Squid proxy server.

Qualsiasi sistema UNIX è in grado di eseguire il routing tra diverse interfaccie di rete così come la redirezione del traffico, ma se Squid va in errore ne risentirà tutto il traffico web che attraversa quel percorso della rete. In dettaglio possiamo affermare che il transparent proxying è un termine piuttosto comune che descrive una metodologia con la quale si posiziona un gateway tra la zona militarizzata (MZ) e la rete degli utenti o zona militarizzata di secondo livello (MZ-L2) per controllare, filtrare e redirezionare il traffico HTTP. In figura 6.3 si riassume schematicamente il concetto appena esplicato.
La rete IP pubblica è la rete di proprietà dell'ISP (Internet Service Provider), il router è la porta di accesso che delimita la rete pubblica dal nostro segmento di rete pubblico. Nella DMZ vengono mappati gli indirizzi pubblici, nella MZ gli indirizzi IP sono quelli riservati alle reti private. Con questo sistema tutti gli accessi di tipo *www* verranno convogliati al gateway, il Transparent Proxy funziona unicamente per le richieste di tipo HTTP. E' possibile paragonare la tecnica del transparent proxy ad uno switch di livello sette (vedi figura 6.2). Si tratta in effetti di una apparecchiatura molto sofisticata che opera al livello sette del modello OSI, quindi lavora sullo strato delle applicazioni. Per operare correttamente questo tipo di apparati hanno la necessità di tradurre gli indirizzi internet e di interpretare il traffico IP.
Il protocollo di trasporto TCP opera però sul quarto livello del modello OSI. Uno switch di livello sette fornisce le stesse opzioni che uno switch di livello quattro offre solo addizionalmente. Nel caso della tecnica del transparent proxy, Squid si installa su una macchina che opera da gateway, questo tipo di configurazione viene utilizzata nelle grandi LAN/WAN dove il numero dei PC client è molto elevato. I pacchetti inviati dai PC client che fanno

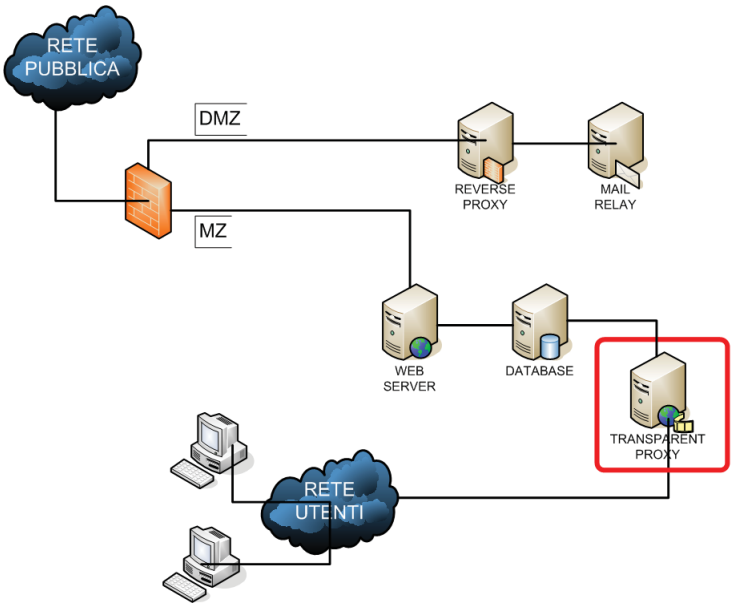

Figura 6.1: Schema transparent proxy.

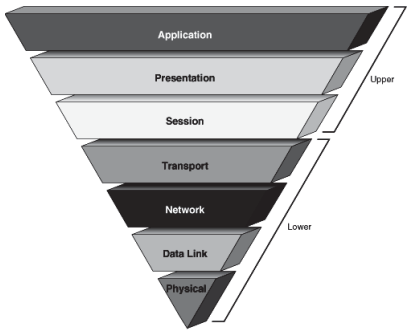

Figura 6.2: Modello ISO-OSI.

richiesta del protocollo HTTP vengono ridirezionati dal gateway o dallo switch di livello quattro, che nel caso specifico è appunto incluso nella nostra macchina Linux o nel nostro sistema BSD dove viene eseguito Squid.
Il processo di ridirezione dei pacchetti non viene effettuato direttamente dal proxy server ma viene processato da una applicazione che si occupa di gestire le regole del traffico IP, questo tipo di applicativo consente di filtrare e manipolare i pacchetti. Quando il Kernel del sistema operativo riceve i pacchetti diretti verso la porta 80 verifica le regole di redirezione ed aggiusta il pacchetto ricevuto cambiandone la porta di destinazione. Se Squid è in ascolto sulla porta 3128 il traffico IP verrà intercettato e rediretto verso la porta 3128 dove normalmente rimane in ascolto Squid. Per effettuare il setup del transparent proxy è necessario effettuare due attività

1. configurare l'applicazione che esegue il filtro dei pacchetti
2. configurare Squid

In figura 6.3 troviamo uno schema esplicativo del funzionamento di uno switch di livello sette.

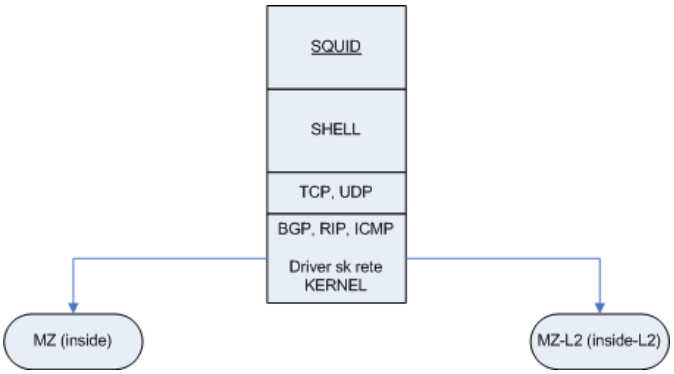

Figura 6.3: Switch di livello sette.

Tutte le richieste provenienti dalla rete mz-l2 e dirette verso la porta 80 dei sistemi che sono oltre il segmento di rete mz, verranno manipolate dalla applicazione che esegue il filtro dei pacchetti e che lavora sia a livello tre che a livello quattro della pila OSI, questi pacchetti vengono ridirezionati ed inviati alla porta 3128 dove normalmente rimane in ascolto Squid che lavora invece al livello sette, ovvero al livello delle applicazioni della pila OSI.

6.2 Filtraggio dei pacchetti:

E' necessario configurare l'applicazione che esegue il filtro dei pacchetti IP per accettare il traffico proveniente da qualsiasi indirizzo e diretto verso la porta 80 e redigerlo verso la porta dove è in ascolto l'applicazione di cache (nel nostro caso la 3128). Per effettuare questa attività sono necessarie delle applicazioni che consentono di gestire funzionalità avanzate come ipfiltering e ipforwarding. In genere, questo tipo di applicazioni vengono fornite con il Kernel del sistema operativo.
I sistemi BSD, utilizzano per eseguire il filtraggio dei pacchetti, **ipfw(8)** e **natd(8)** per abilitare la funzionalità di transparent proxy è necessario compilare (o ricompilare) il Kernel o aggiungere dei moduli che possono essere caricati dinamicamente.

6.3 Il software Squid:

I proxy servers, oltre ad essere utilizzati anche sulla rete Internet, vengono installati soprattutto nelle LAN/WAN private di grandi dimensioni. Come vedremo in seguito, gli amministratori di rete possono trarre grande vantaggio dall'utilizzo di Squid grazie alle sue comprovate capacità di velocizzazione, di controllo sugli accessi, di gestione delle risorse e del suo elevatissimo livello di sicurezza.

6.4 Come lavora Squid:

Un dispositivo di webcache memorizza una copia degli oggetti web e dei dati maggiormente richiesti in uno spazio disco dedicato . Un utente che si trova all'interno di una rete che non fa ricorso all'utilizzo di questi apparati visualizzerà una pagina web eseguendo una richiesta diretta che verrà inoltrata immediatamente al servente remoto. Il giorno successivo lo stesso utente richiederà nuovamente quella pagina web e probabilmente ne

disporrà di una copia memorizzata nella cache del suo browser. Se nello stesso momento più utenti all'interno della medesima rete richiederanno l'accesso alla stessa pagina web, effettueranno ciascuno una connessione al server d'origine utilizzando sempre la stessa banda. L'installazione di una o più device di webcache consente di richiedere il contenuto di un oggetto web una sola volta per poi memorizzarlo sulla cache locale e renderlo disponibile per tutti gli utenti.

6.5 Riduzione delle esigenze di banda:

Quando un device di webcache viene inserito in un network sarà il solo apparato di rete abilitato a contattare i serventi di origine dei dati. In questo contesto l'apparato viene normalmente situato all'interno di una Local Area Network (LAN) e solo gli utenti accreditati all'interno di quella rete potranno accedervi. I dati transiteranno unicamente sulla rete locale senza appesantire le esigenze di banda relative alla connettività Internet ed il maggior traffico dati avverrà tra il browser web utilizzato dagli utenti e l'apparato di webcache stesso.

6.6 Panoramica sul protocollo HTTP:

HTTP è l'acronimo di Hypertext Transfer Protocol [1] e si tratta di un metodo standard per l'invio dei documenti attraverso la rete web. In particolare possiamo definire HTTP come un protocollo di livello sette che viene utilizzato per trasferire gli ipertesti tra gli HTTP server (Apache, Internet Information Server, ...) e gli HTTP Client (Mozilla, Opera, Konqueror, Internet Explorer...). HTTP ricorre al protocollo TCP per trasportare i pacchetti sulla rete, stabilendo dunque una connessione TCP tra il client ed il server. Gli HTTP server sono anche conosciuti come server web e normalmente restano in attesa di richieste sulla porta 80. Gli HTTP client prendono il nome di browser web e per trasferire le informazioni dal server utilizzano una richiesta conforme al protocollo HTTP 1.1 oppure ricorrono ad una richiesta conforme con la vecchia implementazione HTTP 1.0.

[1] HTTP - RFC2616 - http://www.w3.org/Protocols/rfc2616/rfc2616.txt

6.7 I file descriptor:

Quando Squid è fermo in attesa di richieste utilizza in media 24 file descriptor (file di log e comunicazioni tra processi), ogni singola richiesta HTTP determina l'apertura di una media di 8 file descriptor (singola richiesta e connessioni persistenti).
Nell'ipotesi più pessimistica con la quale si prevede l'apertura di 10 o più file descriptors per connessione persistente, con 400 client concorrenti saranno necessari almeno 4096 file descriptor. Una buona configurazione può prevedere 1024 file descriptor, tale impostazione può andare bene se si prevede un traffico medio. La tabella 6.1 può essere utilizzata per dimensionare al meglio il nostro sistema di webcache.
Nel sistema FreeBSD la modifica dei file descriptor si può effettuare senza ricompilare il Kernel. Si impostano i valori corretti al momento dell'avvio del sistema editando il file */etc/sysctl.conf*
I valori scelti nel nostro caso sono pari a:

```
kern.maxfiles=4040
kern.maxfilesperproc=3636
```

File descriptor medi	Utenti correnti	Connessioni persistenti	Totale file descriptor
24	20	20*8=160	160+24=184
24	50	50*8=400	400+24=424
24	100	100*8=800	800+24=824
24	200	200*8=1600	1600+24=1624
24	400	400*8=3200	3200+24=3224
24	500	500*8=4000	4000+24=4024

Tabella 6.1: Tabella file descriptor.

6.8 Requisiti:

Squid utilizza la memoria RAM per tenere una traccia della tabella degli oggetti, un'accesso rapido a questa tabella è fondamentale, il ricorso alla memoria di swap penalizza le prestazioni di Squid sino a renderlo inservibile. Qualsiasi oggetto memorizzato su disco utilizza circa 75 bytes di memoria RAM, la grandezza media di un oggetto è di circa 13 Kb. Utilizzando questi valori è possibile definire uno standard di calcolo in merito al fabbisogno di memoria RAM da parte di Squid.
Ecco alcuni esempi:

- Per 1 GB di disk storage Squid richiederà circa 6 MB di RAM per funzionare correttamente

  ```
  1.000.000/13 = 76.923,07692 oggetti su disco
  75*76.923,07692 = 5.769.230,769 pari a circa 6 MB di RAM
  ```

- Per 8 GB di disk storage Squid richiederà circa 48 MB di RAM per funzionare correttamente

```
8.000.000/13 = 615.384,6154 oggetti su disco
75*615.384,6154 = 46.153.846,16 pari a circa 48 MB di RAM
```

Squid non utilizza in maniera intensiva la CPU, solo al momento dell'avviamento la CPU lavora a pieno regime perchè Squid deve verificare la validità gli oggetti contenuti nella cache. Ne consegue che una CPU di vecchia generazione può penalizzare l'accesso agli oggetti presenti nella cache unicamente nei minuti successivi all'avvio di Squid. Un sistema Pentium 300 Mhz può già essere sufficente per eseguire Squid in piccole realtà aziendali. Per quanto concerne il sottositema dischi lo standard su bus SCSI è la soluzione consigliata, a tale riguardo si rammenta che oggi è possibile utilizzare dei sistemi su bus EIDE o ATA che prevedono il collegamento con delle unità a disco molto veloci, queste unità devono essere in grado di supportare il trasfermento dei dati tramite DMA e devono disporre di una capacità di rotazione pari almeno 7.200 rpm.

Per bilanciare al meglio le performance di Squid è necessario valutare anche altri fattori a riguardo della scelta delle unità a disco.

- numero delle unità a disco
- tipo di file system
- spazio libero su disco
- numero di files presenti sul file system

Capitolo 7

Il programma di interazione con Squid: SquiVi2

7.1 Introduzione:

Il software SquiVi2[1] è un programma di redirezione che si integra perfettamente con Squid. È in grado di filtrare qualsiasi dato in entrata e sono supportati inoltre i tipi *Mime*. I dati scaricati sul server vengono se necessario decompressi con i principali algoritmi di de-/compressione, ed inviati ad uno o più software antivirus presenti sul sistema. Sia i software antivirus che quelli di de-/compressione sono ampiamente configurabili.

Figura 7.1: Screenshot SquiVi2.

SquiVi2 mostra all'utente, in attesa che il file venga scaricato sul proxy ed in seguito scansionato, una pagina di stato come in figura 7.1 e figura 7.2. Una volta scansionato può essere normalmente salvato dal client sul proprio PC.
Questo meccanismo, illustrato meglio in figura 7.3, è il seguente:

1. L'utente effettua la richiesta di scaricare un file da Internet. Questa richiesta viene inoltrata a Squid.

[1] http://squivi2.sourceforge.net

Figura 7.2: Screenshot SquiVi2.

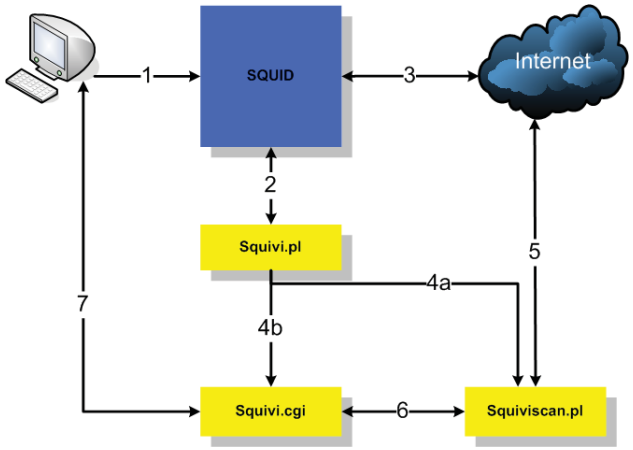

Figura 7.3: Funzionamento di SquiVi2.

2. Se il file può essere scaricato[2], procede la richiesta.

3. Nel caso in cui fosse un file innoquo[3], questo viene scaricato immediatamente dall'utente senza ulteriori passaggi.

4. In caso contrario:

   (a) Viene richiamato lo script *squiviscan.pl* chimato a sua volta da *squivi.cgi* che processa tramite l'antivirus il file.

   (b) L'utente in attesa del termine del *download* ha di fronte una pagina che riporta il progresso dello scaricamento ed il responso dell'antivirus al termine della scansione prodotto dallo script *squivi.cgi*.

5. Lo script *squiviscan.pl* esegue il programma **wget** e scarica il file sul proxy, decomprimendolo se necessario e lo passa all'antivirus.

6. *squivi.cgi* nel frattempo mantiene informato l'utente, e previene così anche un timeout da parte del browser.

[2]Si può parametrizzare qualsiasi vincolo su tipologie di file, siti, ...
[3]File .mpeg, .avi, .mp3 possono essere messi in una sorta di safe list

7. Nel caso in cui il file appena scaricato sia esente da virus, l'utente può scaricare normalmente tale file sul proprio PC.

In figura 7.1 ed in figura 7.2 si può notare quello che appare all'utente durante il periodo di processamento del file da scaricare.
In questo caso specifico il file processato contiene un virus[4].

7.2 Requisiti:

- Perl[5] v5.8 con i seguenti moduli
  - Config::General
  - Data::Dumper
  - Fcntl
  - File::Basename
  - File::Copy
  - File::Path
  - Getopt::Std
  - IO::Handle
  - ICP::Open2
  - LWP::Simple
  - Storable
  - URI::Escape
  - Unix::Syslog
- Squid
- Web server con estensioni CGI
- wget[6]
- ClamAV antivirus
- Programmi di de-/compressione (tar, gunzip, bunzip, unzip, ...)

[4]EICAR ha a disposizione dei file per testare i software antivirus - http://www.eicar.com/anti_virus_test_file.htm
[5]http:/cpan.perl.org
[6]http:/gnu.org/software/wget/wget.html

Capitolo 8

Interazione componenti software

8.1 Modulo antivirus per m0n0wall:

Per l'implementazione di un sistema antivirus sul firewall m0n0 è stato necessario, a scanso di dover riscrivere un software antivirus che lavorasse sul livello 3 del modello OSI, ricorrere all'utilizzo di un software che facesse da proxy. Con questo sistema, e con l'utilizzo di un'altro software che lavora in concomitanza con l'antivirus, è possibile scansionare i file che passano attraverso il firewall.

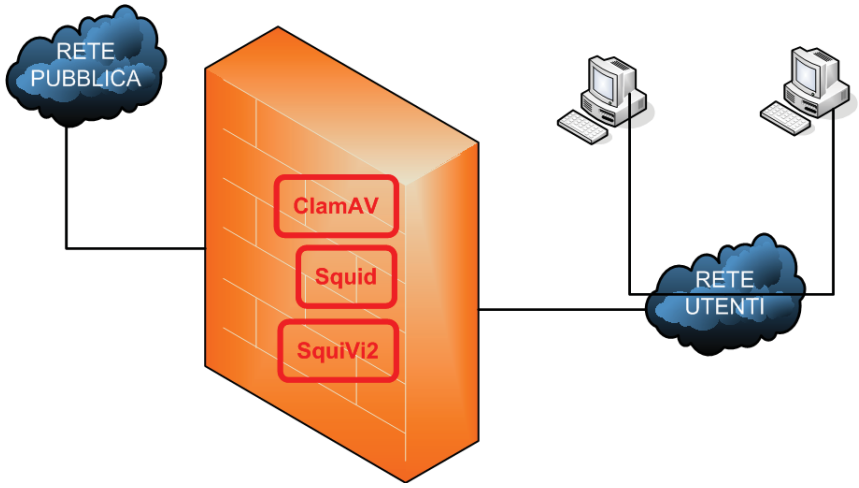

Figura 8.1: Modulo antivirus per m0n0wall.

8.2 Integrazione dei tre sistemi di filtraggio:

Per effettuare la scansione dei file che passano attraverso il file, abbiamo utilizzato tre sistemi uno dei quali è il firewall stesso. Come si può vedere in figura 8.2 il client che effettua la richiesta ad un server, viene in seguito intercettata da Squid in ascolto su una porta dedicata, e con l'aiuto di un web server che è in ascolto sulla porta 80, ridireziona il file richiesto a SquiVi2. Quest'ultimo si occupa di far apparire all'utente una pagina dove viene inizialmente mostrato lo stato del *download*. Questo artificio serve a non incombere in un *timeout* indesiderato del browser. Al termine del download, che viene eseguito e messo in *cache* da Squid, SquiVi2 delega a ClamAV il compito di eseguire una scansione del file per il controllo di eventuali virus.

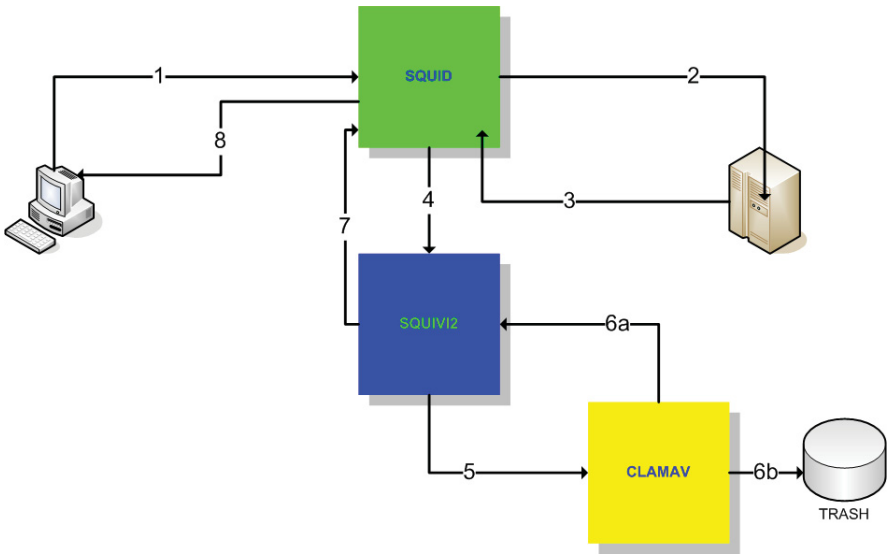

Figura 8.2: Squid, SquiVi2 e ClamAV.

Nel caso in cui quest'ultimo sia esente da virus, all'utente si presenta nella stessa pagina il risultato della scansione stessa, ed in seguito si procede con il *download* e relativo salvataggio del file come abitualmente accade. Nell'eventualità che il file contenga un virus conosciuto, nella schermata creata da SquiVi2 appare il messaggio ritornato dall'antivirus, ed il file non può essere scaricato dal client e viene direttamente cancellato dalla cache di Squid.
Per il funzionamenot di SquiVi2 si veda il capitolo 7.

Capitolo 9

Configurazione ClamAV tramite WebGUI

9.1 Introduzione:

Attraverso l'utilissima funzionalità di m0n0wall, di poter accedere alle configurazioni del firewall tramite WebGui, ho implementato le necessarie pagine in PHP per la configurazione dell'antivirus ClamAV, così come quelle per FreshClam che provvede all'aggiornamento del database di ClamAV.

9.2 Configurazione dell'antivirus:

Le funzionalità personalizzabili di ClamAV sono molteplici, e non possono essere tralasciate. Per rendere maggiormente efficace e parametrizzabile, ho volutamente rappresentato tutte le possibili voci presenti nel file di configurazione. Nel caso in cui si voglia utilizzare una configurazione standard, viene prodotto automaticamente, senza immetere nulla nei campi.

9.3 Configurazione per l'aggiornamento:

Le stesse affermazioni fatte per la configurazione di ClamAV, possono essere riportate per FreshClam. L'amministratore deve avere la possibilità, a seconda delle necessità, di poter intervenire sulle funzionalità di questo importantissimo strumento, senza il quale il nostro antivirus sarebbe inutile: un database aggiornato.

9.4 Logging:

A livello di utente, la rilevazione di un file infetto viene presentata come esposto nel capitolo 7.1 figura 7.1 e figura 7.2. Per quanto riguarda invece il lato dell'amministratore, vi è una sezione nel WebGui appositamente dedicata ai log prodotti da ClamAV, Squid e Squivi. Questo permette di tenere sotto controllo eventuali anomalie dei programmi citati in precedenza attraverso i file di log normalmente prodotto, come mostrato in figura 9.3[1].

[1]Per motivi di sicurezza, tutte le *entry* dei log sono state rese illeggibili.

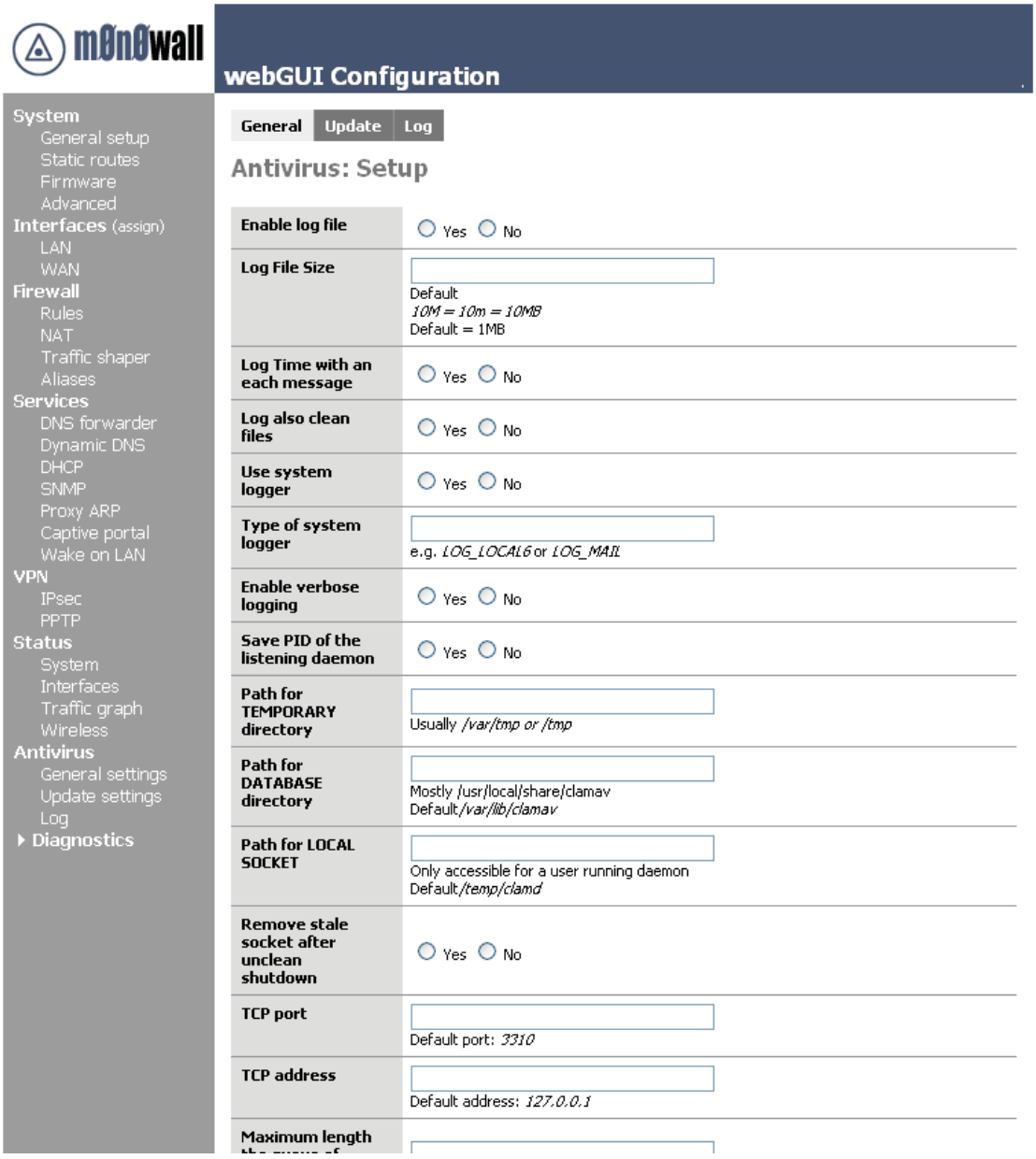

Figura 9.1: WebGui: Configurazione AV.

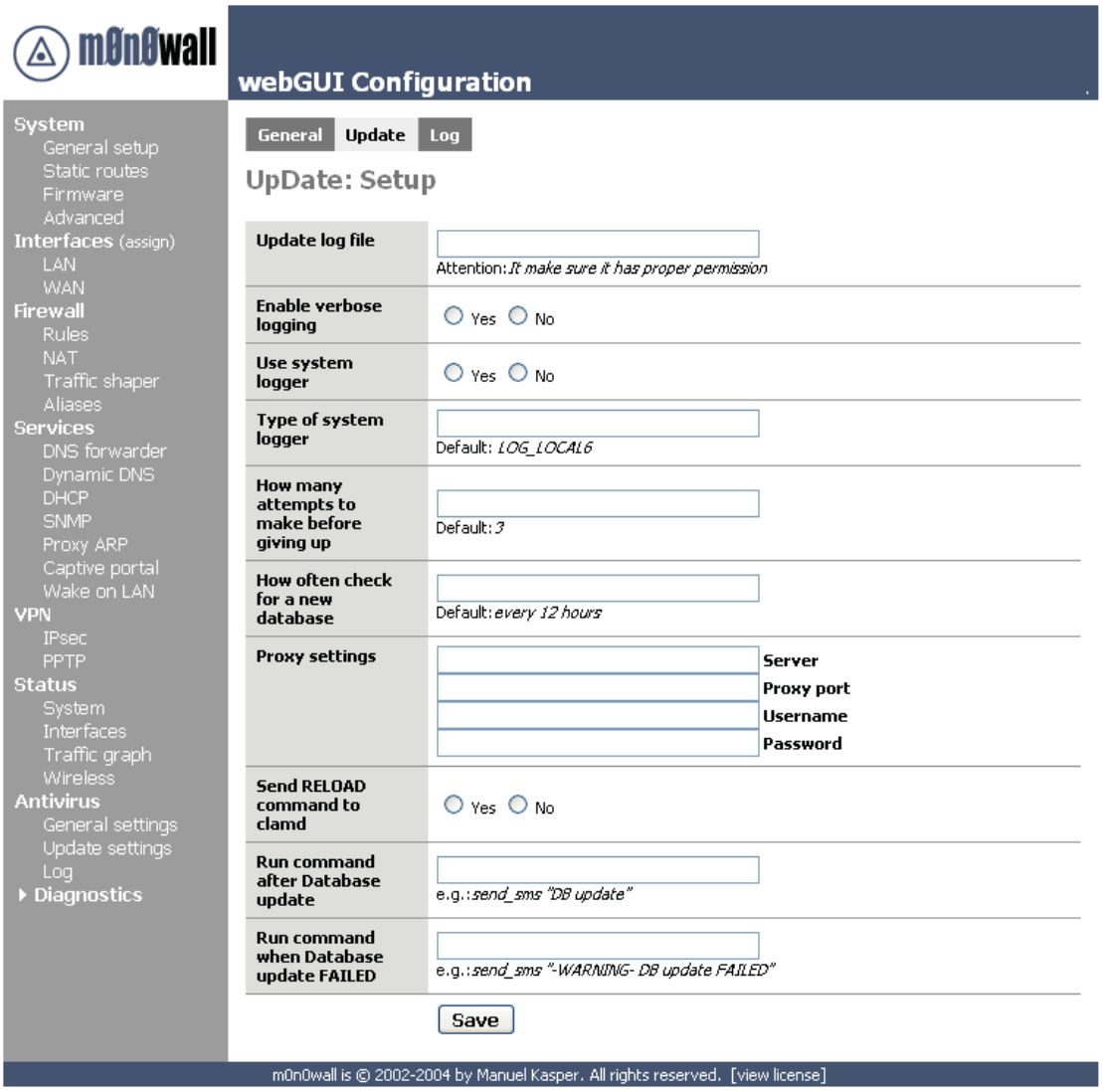

Figura 9.2: WebGui: Configurazione aggiornamento AV.

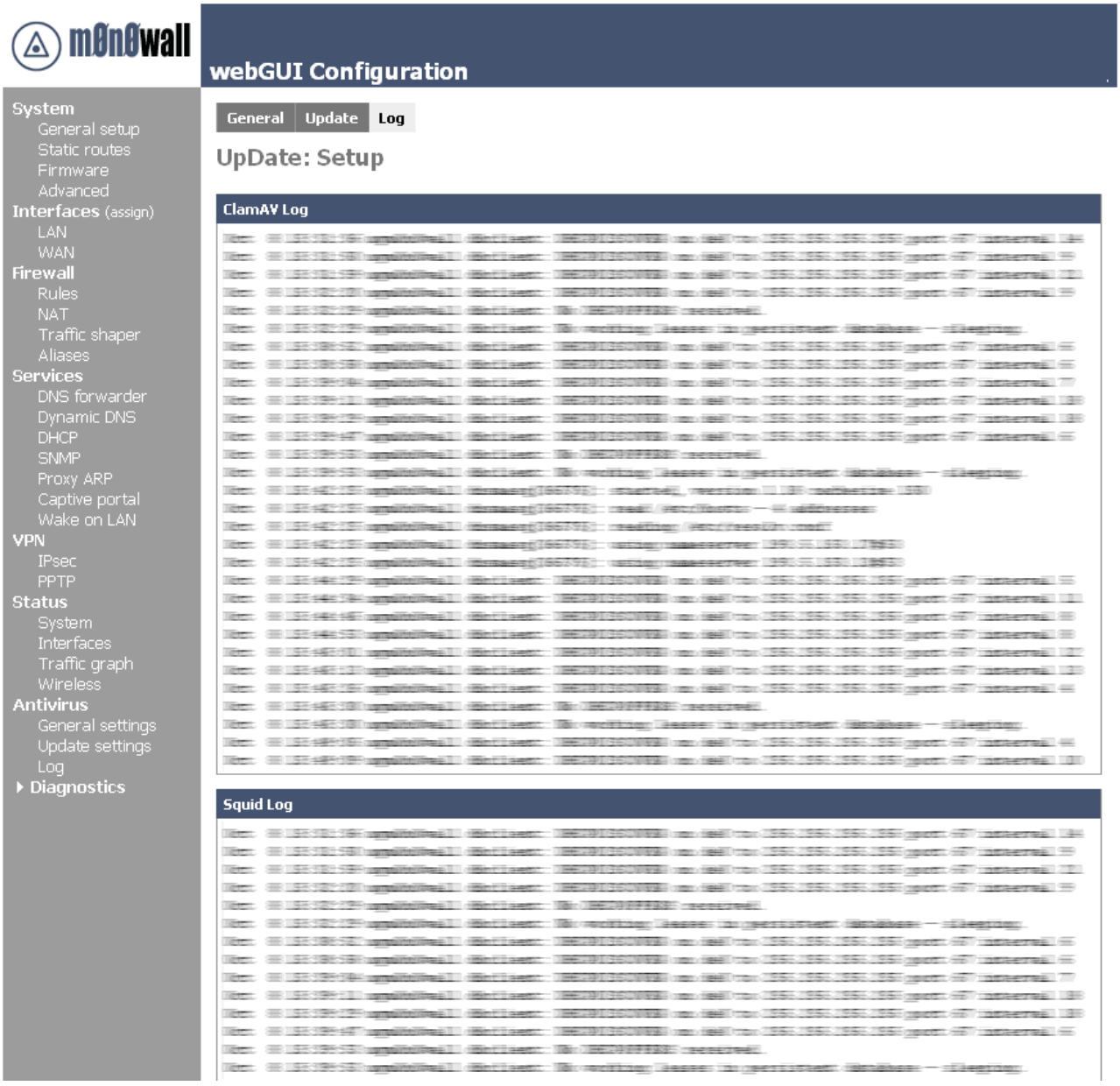

Figura 9.3: WebGui: Logging.

Capitolo 10

Conclusioni

Questo lavoro di diploma mi ha permesso di confrontarmi con diverse problematiche riguardanti la possibilità di integrare un sistema antivirus su un firewall.
In primo luogo, la difficoltà di reperire un software antivirus open source che garantisse la possibilità di aggiornare il database delle signature. Questa ricerca è stata improntata provando varie soluzioni trovate sulla rete. Alcune le ho dovute scartare a priori visto che non permettevano l'aggiornamento (richiesta esplicita per il lavoro di diploma). Una volta trovato il software giusto si è posta la problematica di dover scansionare il traffico: questo di primo acchito può sembrare semplice, ma purtroppo questo tipo di funzionalità è prerogativa di software a pagamento. Questo mi ha obbligato a dover trovare una soluzione per fare in modo di salvare preventivamente i file, poterli scansionare ed infine mandarli al mittente.
Fortunatamente, grazie alle conoscenze acquisite nella mia formazione e grazie alla mia ricerca di documentazione sul web, ho risolto il problema utilizzando un software di web caching. Questo permette di depositare i file provenienti dall'esterno sul proxy.
A questo punto serviva qualcosa che facesse da mediatore tra il sistema antivirus ed il proxy. La ricerca è stata difficoltosa visto che molti sistemi prevedono di scansionare i file direttamente sul client e non su di un firewall. La soluzione si è presentata con un software scritto in parte in Perl ed in parte in CGI che permette di utilizzare vari tipi di antivirus, completamente parametrizzabile secondo le esigenze, ed infine il fattore più importante: open source.
La fase intermedia consisteva quindi nel far interagire i tre meccanismi, in maniera ottimale, testando le varie situazioni che si potevano presentare.
Una volta terminata questa fase, ho dovuto procedere con il *porting* di questo sistema sul firewall *m0n0wall*.
Questa fase è stata la più critica, visto che il firewall è stato implementato su una versione minimale di un sistema FreeBSD. Molti componenti che su di un'installazione normale sono presenti, su m0n0 non ci sono. Questo mi ha portato ad effettuare due tipologie di approccio:

10.1 Portare il software dentro m0n0wall:

In questa fase, ho utilizzato la *root* del firewall introducendo il software necessario. Una volta controllate le dipendenze di ogni elemento, ho proceduto a ricreare la situazione come si presentava sul mio computer di test. A questo punto, procedendo con i test, ho

incontrato molti problemi, alcuni dei quali erano dovuti ad elementi mancanti sul sistema operativo *bare-bone* di m0n0wall. Inoltre il software antivirus richiedeva un utente non privilegiato per eseguire la scansione dei file. Lo stesso discorso valeva per il software di web caching. Questo ha creato notevoli problemi visto che sul firewall la gestione degli utenti non è demandata come al solito a database di utenti (leggi *passwd*, *master.passwd*, oppure *group*, ma direttamente dalla configurazione del firewall stesso.

10.2 Considerazioni finali:

A mio avviso m0n0wall è un'ottima base di partenza. Questo firewall è stato progettato veramente ad hoc, ma per questo genere di estensioni, richiede una profonda modifica di alcune logiche di implementazione. Sono convinto che Manuel Kasper abbia progettato m0n0wall in maniera encomiabile, ma non ha tutt'ora pubblicato una documentezione utile a capire tutte le sue scelte riguardo le varie implementazioni. Questo ha portato il sottoscritto a dover smontare effettivamente il firewall per studiare tutti questi aspetti, dovendo però accorgermi che alcuni approcci limitano di molto la possibilità di aggiunte così corpose.
Queste considerazioni personali sono dettate anche dal fatto che nel mondo *open source* non si trovano ancora soluzioni adatte a questo tipo di funzionalità richieste dal progetto (aggiornabili, e che lavorano a livello molto basso del modello OSI).
Come detto in precedenza, per concludere, credo che m0nowall si possa utilizzare come base, utilizzando un sistema *Unix like* e progettando preventivamente, cosa includere a livello di funzionalità.

Capitolo 11

Problemi aperti e possibili sviluppi

11.1 Problemi aperti:

- Risolvere gli errori di librerie di Perl.

11.2 Possibili sviluppi:

- Maggiore integrazione con il firewall m0n0wall: le regole sono da integrare con il sistema di transparent proxy.
- Possibilità di utilizzare un utente con privilegi ridotti all'interno della configurazione del web server. Al momento infatti è possibile unicamente utilizzare l'utente root di default per accedere alla WebGUI del firewall.
- Pagina riassuntiva con le statistiche dei rilevamenti dell'antivirus. L'utilizzo di *JpGraph*[1], che avevo preso in considerazione inizialmente, ma che purtroppo per mancanza di tempo non ho potuto integrare nella GUI.

[1] http://www.aditus.nu/jpgraph/

Capitolo 12

Bibliografia

Siti Internet

- m0n0wall Home Page
  `http://www.m0n0.ch/wall`
- Clam Antivirus Home Page
  `http://www.clamav.net`
- Squid Home Page
  `http://www.squid-cache.org`
- SquiVi2 Home Page
  `http://squivi2.sourceforge.net`
- Apache Home Page
  `http://www.apache.org/`
- Sourceforge
  `http://www.sourceforge.net`
- FreeBSD Home Page
  `http://www.freebsd.org`
- Michael Iedema Home Page
  `http://michael-i.com`
- PHP Home Page
  `http://www.php.net`
- Google
  `http://www.google.ch`

Capitolo 13

Allegati

Allegati documentazione

- Listato pagina PHP per la configurazione dell'antivirus
- Listato pagina PHP per la configurazione degli aggiornamenti dell'antivirus
- Listato pagina PHP per la visualizzazione dei log

Allegati CD-ROM

- Documentazione progetto (monoAV.pdf)
- Sorgenti documentazione LaTeX
- Presentazione PowerPoint (PresentazioneAV.ppt)
- Immagine m0n0wall con modulo antivirus
- Pagina PHP per la configurazione dell'antivirus (antivirus.php)
- Pagina PHP per la configurazione degli aggiornamenti dell'antivirus (antivirusu.php)
- Pagina PHP per la visualizzazione dei log (antivirusl.php)
- Documentazione ufficiale Clam Antivirus v0.80 (clamdoc.pdf)
- Documentazione ufficiale SquiVi2 (squivi2.pdf)

```
Nota: La parte di documentazione maggiormente tecnica, è stata consegnata al
      committente su sua esplicita richiesta, e non figura in questo rapporto.
```