Whoml

(What's happening on my lan)

Progetto SGR 2009 di Cappelli Federico (253215)

Descrizione generale

Lo scopo del progetto Whoml è quello di fornire ai non addetti ai lavori uno strumento facile e intuitivo per comprendere cosa sta succedendo all'interno della propria rete Lan, che si collochi tra Wireshark, tool troppo complicato e tecnico e la miriade di applicazioni che si curano di un aspetto della rete, mostrando una serie di informazioni (da ampliare con le versioni successive) su gli host collegati e sulla loro attività.

Info host: nome, IP, mac, data ultima volta visto attivo, siti web visitati, user agent, ultimi protocolli di livello di rete utilizzati, ultimi protocolli di livello applicazione utilizzati.

L'interfaccia è web based e mostra gli host con le loro informazioni, il link del mac porta ad un web site che fornisce un servizio di identificazione del produttore dell'interfaccia di rete attraverso il mac, e i grafici RRD sull'utilizzo della rete.

Compilazione

Macchina di produzione: Apple Macbook Snow Leopard, compilatore g++ v. 4.2

Librerie di terze parti:

- Libpcap 1.0.0
- Boost 1.40.0
- mongoose (code.google.com/p/mongoose)

Unix Tool:

- netstat
- fping 2.4b2
- arp
- nslookup
- nmblookup 3.0.28a-apple

Utilizzo

- Per lanciare l'eseguibile:
  whoml -i <interface> [-w] [-v]

  -i <interface> | Interfaccia di ascolto

  -w | abilita il webserver

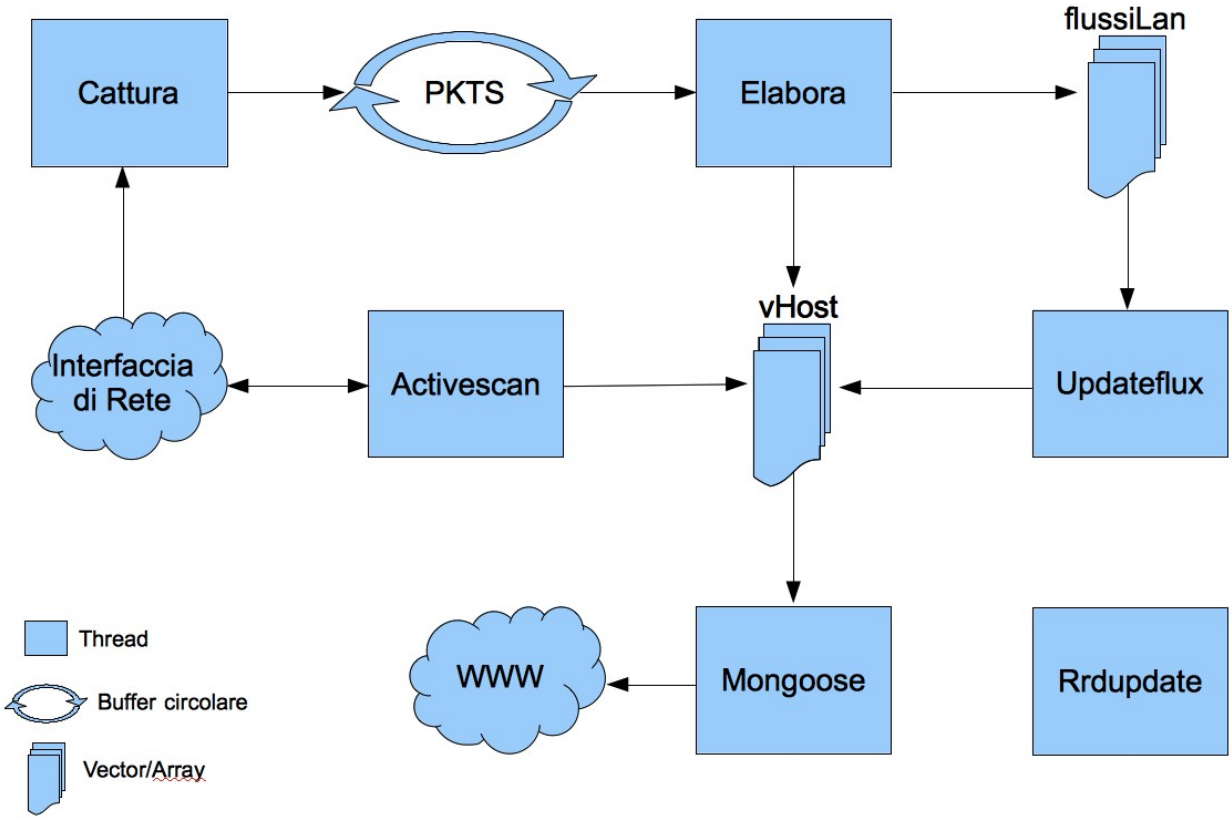
  -v Modalità verbosa

  Esempio: whoml -i eth0 -w -v
- Per fermare l'esecuzione:
  da shell: “ctrl + c” .

da interfaccia web: cliccare sul link “termina”.

- Interfaccia su http://127.0.0.1:8080

Schema esplicativo



Scelte progettuali

Thread:

- Cattura: Il thread cattura i pacchetti di rete tramite la libreria pcap e li inserisce in un buffer circolare, inoltre raccoglie i dati necessari alla creazione dei grafici RRD.
- Elabora: Thread principale del progetto che recupera i pacchetti dal buffer circolare e li analizza uno a uno, in prima istanza analizza il contenuto del pacchetto, in ordine IP-TCP/UDP, memorizzando i vari protocolli usati, nel caso trovi TCP analizza l'http e se contiene una GET memorizza host contattato e user agent. In seconda istanza Identifica il mittente e il destinatario del pacchetto, se non sono già presenti nel buffer circolare degli host (vHost) li aggiunge, altrimenti inserisce i dati raccolti nei rispettivi host.

  Altra attività importante è quella di memorizzare i flussi di rete nell'apposita struttura dati (vedere spiegazione successiva) e nel caso di flusso TCP che riceva un syn cancellare il relativo flusso.

  Ho valutato anche la possibilità di lanciare n thread di tipo cattura sul buffer circolare ma facendo dei test sulle reti wifi più grandi che ho trovato ho notato che un solo thread era più che sufficiente per smaltire i pacchetti catturati.
- Activescan: Scansiona attivamente la rete ad intervalli di 20 secondi lanciando un ping su

tutti gli ip dell'intervallo (fping) alla ricerca degli host attivi in modo da capire quando un host sparisce dalla rete o quando appaiono nuovi host che pero' non generano traffico. Tenta anche di risolvere il loro nome con diverse tecniche (nmblookup, nslookup, gethostbyaddr).

- UpdateFlux: Scansiona la struttura dati “flussiLan” e aggiorna i protocolli usati degli host presenti nella struttura dati principale, segna come inattivi gli host che non hanno registrato attività per più di 5 minuti ed elimina i flussi inattivi da più di 5 minuti.
- RrdUpdate: Thread che si occupa della creazione e dell'aggiornamento dei database RRD (byte totali visti sull'interfaccia di rete e quantità di pacchetti TCP e UDP nel totale del traffico), oltre che alla creazione delle immagini dei database.

Strutture dati:

- vHost: Vettore di oggetti “host”, questa struttura essendo il perno dell'intero programma è inadeguata, in passi successivi dello sviluppo dell'applicazione sarà sostituita con una hash table.
- flussiLan: E' un array di struct “flusso” (ip mittente, ip destinatario, porta mittente, porta destinatario, numero pacchetti, protocollo, ultimo aggiornamento, terminato) di dimensione “nFlussi”, i flussi sono scritti e letti tramite una funzione hash:

  (ipM+ ipD + pM + pD + prot) % nFlussi

  I flussi TCP sono facilmente individuabili e memorizzabili, al contrario UDP essendo connectionless non ha dei veri e propri flussi, quindi ho assunto che flussi UDP con lo stesso hash siano lo stesso flusso, l'assunto nel caso avessi bisogno della sicurezza sui flussi memorizzati sarebbe sbagliato ma visto che i flussi sono usati solamente per per individuare i protocolli usati dagli host posso permettermi questa approssimazione. Allo stesso modo i flussi UDP vengono marchiati come inattivi solo allo scadere di un tempo prefissato.

  Altro approssimazione “spinta” è stata fatta per l'individuazione dei protocolli, visto che la strada del packet inspection è impossibile per ovvi motivi ho assunto che la porta identificata come porta destinazione nel flusso sia la porta che identifica il protocollo, quindi con uno comando bash ho parsato il file */etc/services* creando uno switch per la conversione porta-protocollo:

  awk 'BEGIN { FS= " " } ($1!="#") { print $1 " " $2 }' /etc/services | cut -d "/" -f1 | awk '{ print $2 " " $1 } ' |sort -u | sort -n | awk '{ printf "\ncase %s:\nprot = #%s#;\nbreak;\n", $1,$2}' >> portToProtocoll.txt

  che poi è stato ulteriormente ampliato con range di porte conosciute.

  Ovviamente la tecnica non è esatta essendo */etc/services* limitato alle porte assegnate ufficialmente e non essendo per definizione una porta associata ad un protocollo, ma sperimentalmente si è rilevata abbastanza attendibile.

Interfaccia web:

L'interfaccia web è fornita da un web server embedded (code.google.com/p/mongoose) tramite l'overloading delle callback, il thread del web server fornisce risposte alle GET:

- /json : risponde con una stringa json (www.json.org) contenente l'intero vettore “vHost”
- /vhost : (debug) fornisce la stringa json ma con formattazione html
- /stop : termina il programma.

La pagina web vera e propria usa html, css e javascript per il parsing della stringa json (nativo del javascript).

Conclusioni

Il progetto ha molte lacune, come l'ottimizzazione delle strutture dati, e ancora molte future che mi ero prefissato mancanti, come ad esempio:

- Individuazione del sistema operativo con metodo passivo
- Implementazione di un viewer SNMP basilare che mostri i dati delle macchine
- Uso del protocollo bonjour per ottenere informazioni dagli host

L'interfaccia definitiva verrà implementata successivamente (esame di Costruzione di Interfacce)