

# Implementation of an Application for Intrusion Detection on Network Virtual Machine

Tesi di Laurea di  
Giorgio Giuseppe Moscardi

Relatori:  
Mario Baldi, Fulvio Risso

Aprile 2007

Al giorno d'oggi, i sistemi per il rilevamento delle intrusioni, comunemente detti *IDS*, dall'inglese *Intrusion Detection System*, costituiscono un componente essenziale delle reti di computer. Infatti, la diffusione delle connessioni ad Internet causa un continuo aumento del numero di utenti e servizi, con il risultato che la quantità di lavoro che deve essere svolto da un IDS aumenta giorno dopo giorno, e gli IDS devono migliorare le proprie prestazioni altrettanto costantemente. Dato che siamo ormai arrivati ad un punto in cui è difficile migliorare ulteriormente questi programmi per quanto riguarda il software, gli ultimi sforzi nel campo sono stati orientati verso l'implementazione in hardware degli IDS, o almeno delle loro parti computazionalmente più intensive. Il passaggio non è scontato, dato che il riutilizzo di codice preesistente non è facilmente realizzabile, a causa delle enormi diversità che presenta l'hardware in commercio, sia per quanto riguarda l'architettura, che per quanto riguarda il metodo di programmazione. Spesso, la soluzione più pratica è quella di ricominciare il lavoro da capo, "reinventando la ruota" per ogni dispositivo.

Il framework NetVM, sviluppato al Politecnico di Torino, ha come scopo la risoluzione di questo problema. Attraverso l'astrazione di un *network processor* e la definizione di un set di istruzioni molto specifico, orientato al processamento di pacchetti, si propone di ottenere la portabilità del codice su un buon numero di dispositivi hardware e software, pur mantenendo alta l'efficienza.

## I sistemi per il rilevamento delle intrusioni

I sistemi per il rilevamento delle intrusioni cercano di rilevare attività anomale all'interno di una rete di computer, spesso effettuate guadagnando l'accesso a un sistema remoto attraverso Internet. Tali attività possono essere di diversi tipi, ma mirano tutte, sostanzialmente, a guadagnare il controllo parziale o completo della macchina o allo scaricamento di dati riservati.

Per gli scopi di questa tesi, l'attenzione sarà focalizzata sugli IDS di rete, detti *NIDS*, *Network IDS*. Tali sistemi lavorano all'interno di una rete, catturando il traffico attraverso un hub o dispositivi simili. In questo modo, essi riescono a tenere sotto controllo tutti gli host della rete, analizzando il traffico destinato a ciascuno di essi e conoscendo i servizi da loro erogati.

### Snort

Snort è un'implementazione software di un IDS di rete passivo. A quasi un decennio dalla sua prima pubblicazione, Snort è diventato lo standard di fatto per il rilevamento delle intrusioni. Possiede numerose funzionalità, ed è in grado di effettuare l'analisi del traffico IP in tempo reale attraverso la ricerca di contenuti specifici, cosa che permette di rilevare quasi tutte le tipologie di attacco.

### Le regole di Snort

Sostanzialmente, Snort funziona attraverso *regole*. Esse specificano una serie di test da eseguire su ogni pacchetto, e un'azione da intraprendere qualora tutti i test diano risultato positivo.

Logicamente, le regole sono divise in due parti: l'*intestazione* e le *opzioni*. Vediamo un esempio:

```
log tcp any any -> 10.1.1.0/24 80 (content: "GET"; msg: "Webserver access");
```

L'intestazione è costituita dalla parte prima delle parentesi. Essa contiene l'azione da effettuare nel caso che la regola sia soddisfatta da un pacchetto, il protocollo e, infine, gli indirizzi e le porte sorgente e destinazione che devono essere presenti nel pacchetto. Le opzioni, invece, sono racchiuse tra le parentesi, e specificano una serie di test da effettuare sulle varie parti del pacchetto, al fine di verificare completamente la regola, oltre che i parametri per contestualizzare gli eventuali avvisi generati.

La versione corrente di Snort supporta diverse azioni, ma le più comunemente utilizzate sono *alert*, utilizzata per inviare un avviso diretto alla console, *log*, che invece effettua semplicemente il logging del pacchetto, e *pass*, che fa sì che il pacchetto sia ignorato. Quanto ai protocolli, Snort supporta quelli comunemente utilizzati su Internet, ossia *tcp*, *udp* e *icmp*, mentre per indirizzi e porte sono supportate le sintassi comunemente utilizzate, inclusa la notazione *CIDR* per specificare un'intera sottorete IP. Sia l'indirizzo che la porta possono essere specificati come *any*, col risultato che ogni indirizzo/porta soddisferà il parametro.

## NetVM

Il framework *Network Virtual Machine (NetVM)* nasce dall'idea di ottenere la portabilità e l'efficienza delle applicazioni che effettuano processamento intensivo di pacchetti su diverse piattaforme software e hardware, attraverso l'astrazione delle piattaforme stesse. Tale idea è la naturale estensione del concetto di "macchina virtuale" che si è già affermato nel mondo dei processori *general purpose*, mediante implementazioni come la *JVM* di *Sun Microsystems*, che rappresentano un ambiente ideale per l'esecuzione di applicazioni generiche, ma risultano poco adatte al processamento di pacchetti di rete. Infatti, le operazioni eseguite da quest'ultima classe di applicazioni sono piuttosto ristrette e molto specifiche (ad es. shift logici, operazioni bit a bit) e una macchina virtuale che le supporti nativamente offrirebbe sicuramente prestazioni migliori per queste. Di conseguenza, astruendo l'architettura di un network processor, piuttosto che quella di un processore general-purpose, si dovrebbe raggiungere un'ottima efficienza sia sui comuni PC che sui network processor reali.

## Il problema

All'inizio di questa tesi, la NetVM si trovava ad uno stadio in cui necessitava un testing severo e, soprattutto, la validazione della sua architettura. Questo poteva essere ottenuto mediante l'implementazione di un'applicazione complessa, che dimostrasse, da un lato, l'effettiva usabilità del framework e, da un altro, la sua efficienza. Si è quindi pensato che un clone di Snort fosse l'applicazione ideale per questi scopi: infatti, Snort esegue molte operazioni sui campi delle intestazioni dei protocolli contenuti nei pacchetti, per svolgere il suo compito, e trova quindi nella NetVM il suo ambiente di esecuzione ideale. Inoltre, esegue anche altri test meno specifici, che hanno permesso di valutare quanto bene la NetVM si adatti a compiti per i quali non è stata espressamente progettata. Ovviamente, perché un IDS funzioni bene, è necessario che le sue prestazioni siano elevate, ed è quindi possibile verificare anche l'efficienza globale del framework.

## Scelte progettuali

La maggior parte delle scelte progettuali sono guidate dalla necessità di raggiungere prestazioni elevate, dato che le analisi di un IDS devono essere compiute in tempo reale. La scelta fondamentale è stata quella di realizzare inizialmente un IDS dalle caratteristiche minime necessarie a raggiungere gli scopi preposti, piuttosto che un clone completo di Snort, a causa della probabile necessità di modifiche al framework NetVM stesso, cosa che avrebbe sdoppiato il lavoro su due fronti. Tuttavia, l'architettura proposta rimane estensibile, rendendo possibile la clonazione completa di Snort con lavori futuri. Un'altra idea cardine del lavoro è quella di non sentirsi limitati dall'implementazione attuale di NetVM, ma di essere pronti a cambiarla e a estenderla.

## Implementazione

L'architettura scelta è stata fortemente influenzata dall'approccio della suddivisione del lavoro in vari moduli che eseguono compiti semplici seguito da NetVM. I moduli sono stati identificati con i vari protocolli potenzialmente presenti nei pacchetti. Dato che un pacchetto, normalmente, contiene un unico protocollo di livello rete (IPv4 o IPv6) e un unico protocollo di livello trasporto (TCP, UDP o ICMP), le interconnessioni tra i moduli sono state strutturate in modo da creare una struttura a quattro livelli, come mostrato in figura 1.

Il flusso di esecuzione è il seguente: il primo passo consiste nella lettura dei file di configurazione in formato Snort, con la conseguente memorizzazione delle regole in memoria. Segue quindi l'inizializzazione della NetVM, con la creazione di tutti i NetPE e la generazione del codice per ciascuno di essi, attraverso la compilazione delle regole.

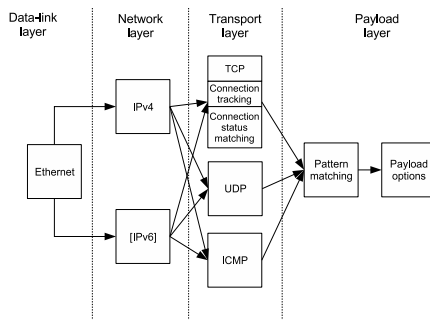


Figura 1: L'architettura di NetVM-Snort

Tabella 1: Risultati dei test

	<b>Snort</b>	<b>NetVMSnort</b>
Pacchetti totali processati	10000000	10000000
Pacchetti TCP	8439040	8439040
Pacchetti UDP	1505540	1505540
Pacchetti ICMP	53429	53429
Alerts	393	393
Pacchetti processati al secondo	97.002,32	6389,59
Memoria totale utilizzata	80240 KiB	8592 KiB
Memoria utilizzata per string-matching	32.84 MiB	1028.94 KiB

A questo punto, con l'inizializzazione di libpcap si completa la fase di preparazione, e si passa al ciclo principale del programma, in cui i pacchetti vengono catturati dalla rete e inviati alla NetVM, che li processa, fornendo l'elenco delle regole verificate. L'elenco viene infine elaborato da un modulo di uscita, che intraprende le azioni specificate nelle regole.

## Modifiche a NetVM

Nel corso del lavoro, alcuni punti del framework NetVM sono stati ritenuti passibili di miglioramento, e si sono quindi proposte ed implementate alcune modifiche. In particolare, è ora possibile associare dati arbitrari ad un pacchetto, cosa che permette all'applicazione di tenere traccia delle regole che il pacchetto soddisfa. Inoltre, sono state implementate alcune parti ancora mancanti delle specifiche: ora il framework NetVM comprende due tipi di coprocessori, utilizzati per implementare i moduli di connection tracking e string-matching.

## Testing

La parte finale del lavoro consiste nel testing delle prestazioni dell'applicazione, confrontandole con quelle di Snort. I risultati dei test sono riassunti nella tabella 1.

A parità di hardware, parametri di compilazione, regole e traffico analizzato, Snort si è rivelato in grado di processare il traffico 15 volte più velocemente. Si tratta di un risultato atteso, per diverse ragioni. Il motivo principale è rappresentato dalla scarsa qualità dell'implementazione dell'interprete del bytecode, che è largamente passibile di miglioramenti. Purtroppo non è stato possibile utilizzare il compilatore Just-In-Time, a causa delle modifiche effettuate al framework, che richiedono l'implementazione delle istruzioni aggiunte. Vi sono, inoltre, punti in cui l'applicazione è migliorabile. Il componente di string-matching, invece, si è rivelato sufficientemente rapido, soprattutto alla luce della ridotta occupazione di memoria in confronto a quella dell'automa utilizzato da Snort, che risulta approssimativamente 32 volte maggiore.

Le prestazioni ottenute sono, dunque, rassicuranti. Questo sarà comprovato da ulteriori lavori, già in corso, mirati a realizzare la prima implementazione del framework NetVM su un vero network processor, alla riscrittura del compilatore JIT e all'ottimizzazione dell'interprete.