UNIVERSITÀ DEGLI STUDI DI BERGAMO

FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica

Classe n. 9

UN NUOVO MODELLO DI SISTEMA PER L'IDENTIFICAZIONE DELLE INTRUSIONI INFORMATICHE: IL ROUTER-IDS

Relatore Stefano Paraboschi

Prova finale di Marco Balduzzi

NOME COGNOME

Matricola n. 33248

ANNO ACCADEMICO 2003/2004

Questa pagina è stata lasciata intenzionalmente bianca.

Licenza

Indice

0. Sommario

Questa tesi propone un modello innovativo di Intrusion Detection System (software per il rilevamento delle intrusioni informatiche) basato sull'approccio context-based. Tre principi stanno alla base dell'approccio utilizzato: focus, conoscenza a priori e correlazione.

La prima parte del documento introduce gli obiettivi della sicurezza informatica, espressi in termini di confidenzialità, integrità e disponibilità dei dati (paradigma C.I.D.).

Seguiranno due capitoli dedicati ai motivi che hanno spinto la comunità di ricerca nel campo dell'Information Security allo studio di un approccio alternativo a quello classico del "Who are you? What can you do?". Sarà illustrato il concetto di Intrusion Detection System e verrà descritto lo stato dell'arte di questa tecnologia fornendo alcune classificazioni rispetto all'approccio utilizzato nell'individuazione delle intrusioni, alle caratteristiche e al tipo di sorgente dati gestita dell'IDS (questa ultima proprietà ne determina la famiglia di appartenenza).

Il quarto capitolo elenca le soluzioni Opensource più diffuse di Intrusion Detection System.

Il quinto capitolo evidenzia i limiti degli attuali sistemi Intrusion Detection quali l'alto numero di falsi negativi e positivi, la mancanza di valorizzazione delle informazioni e la limitata visione di contesto.

La seconda parte del documento si apre con la descrizione dei tre principi dell'approccio context-based: per "focus" si intende la capacità dell'IDS di adattare il proprio comportamento a contesti di tipo e dimensione diversa; grazie alla "conoscenza a priori" la configurazione dell'IDS è modellata secondo la tipologia dell'infrastruttura di rete, del contesto operativo e dei rischi a cui è esporto il sistema monitorato; la correlazione tra eventi provenienti da più IDS arricchisce il contenuto informativo sintetizzando ricche informazioni sulla rilevazione delle intrusioni e riducendo il numero di falsi positivi/negativi.

Il settimo capitolo descrive il modello proposto presentandone una semplice implementazione software, sufficiente ad evidenziare le caratteristiche innovative e migliorative dell'approccio context-based. Il codice sorgente del programma è stato inserito nel capitolo successivo.

La tesi si conclude con la bibliografia e i riferimenti.

1. Sicurezza dell'informazione: dall'approccio tradizionale a quello moderno

Gli obiettivi della sicurezza dell'informazione sono descritti in termini di confidenzialità, integrità e disponibilità. Questo modello prende il nome di **"paradigma C.I.D."** (o usando i termini inglesi confidentiality, integrity e availability, paradigma C.I.A).
La confidenzialità è la capacità di un sistema di offrire i propri servizi soltanto a chi ne ha l'autorità per farlo. Possiamo rappresentare formalmente questo attributo come una relazione di lettura tra chi usa il sistema e i servizi offerti da quest'ultimo. Per integrità si definisce la capacità di un sistema di rendere possibile solo alle persone autorizzate la modifica delle sue risorse e dei suoi dati, in modo da mantenere una consistenza tra questi dati e le funzioni svolte dal sistema. Infine il termine disponibilità indica la capacità del sistema informatico di offrire sicuramente, tempestivamente e in ogni circostanza l'accesso alle persone che possiedono il diritto di farlo.

Qualsiasi applicazione che offre un accesso multi utente richiede un livello di sicurezza minimo per garantire, quanto possibile, il paradigma "C.I.D.". L'implementazione tradizionale di un sistema di sicurezza consiste nell'associazione utente–privilegio secondo il **"paradigma identificazione e gestione dei privilegi"**, solitamente riassunto dal motto *"Who are you? What can you do?"*: il sistema chiede all'utente (che può essere una persona o un altro sistema) di identificarsi e usa questa identità per attivare una serie di regole operative. Il successo di questo paradigma sta nella sua ortogonalità rispetto al problema: non è infatti difficile applicare il modello di login e autenticazione ad un sistema esistente di qualsiasi tipo.

Tuttavia il paradigma C.I.D., nella sua linearità e semplicità, rappresenta un "mondo ideale" impossibile da garantire certamente: un sistema informatico non potrà mai essere considerato perfettamente sicuro e nell'implementazione di un sistema di sicurezza è bene saperlo fin dall'inizio. Nessuna politica di sicurezza applicata al sistema potrà mai impedire che prima o poi si verifichi una intrusione, più o meno grave.
I sistemi reali usati nelle infrastrutture odierne presentano problematiche di non facile

soluzione. Il bisogno sempre maggiore di sicurezza si scontra con la necessità di funzionalità del sistema stesso. Spesso ci si trova a dover giungere a un compromesso tra funzionalità e sicurezza. Sarebbe troppo facile garantire un livello di sicurezza certo restringendo l'accesso al sistema al punto tale da negare il servizio anche alle persone autorizzate. In tal caso si andrebbe soltanto contro il principio di disponibilità del paradigma C.I.D.!
Come se ciò non bastasse è pratica comune per le aziende comprare il software sotto forma di COTS (Commercial, Off The Shelves) dal momento che mettersi a produrlo internamente comporterebbe un onere troppo alto da sopportare. Se questa pratica ha aspetti economici positivi, d'altra parte è una vera e propria minaccia per la sicurezza dell'azienda. Un software di terze parti rilasciato sotto copyright impedisce all'acquirente qualsiasi forma di controllo del codice, e lo obbliga ad affidare la propria produttività a un software spesso poco testato e lontano dai requisiti di sicurezza minimi. Inoltre l'esperto di sicurezza si trova ad applicare delle policy a un software che non può manipolare direttamente. Per lui diviene necessario aggiungere componenti security-releated a quelli esistenti che spesso sono di difficile integrazione e interoperabilità. E' difficile pensare che un aggregato di componenti talmente eterogenei rispetti le specifiche di sicurezza definite a priori.

L'approccio tradizionale al paradigma “C.I.D.” pone le sue fondamenta sul concetto di password (e di token). Entrambi questi metodi di identificazione sono da considerarsi deboli. La scelta comune di utilizzare password corti e facili da ricordare permette ad un attaccante di recuperarle in poco tempo tramite un *attacco a dizionario*. Anche il vizio degli utenti di scrivere la propria password in posti visibili, come sul monitor del PC o su un post-it appeso sull'armadio dell'ufficio, non aiutano di certo a prevenire un furto di identità. Così come le password anche i token, benché entità fisiche “uniche”, possono essere rubati, persi o falsificati.

A questo punto sono evidenti i limiti dell'approccio tradizionale al paradigma C.I.D., quello del *“Who are you? What can you do?”* basato sulla relazione risorsa-utilizzatore.
Il **concetto di Intrusion Detection System** (IDS) è complementare al paradigma tradizionale e pone il suo fondamento sulla constatazione che non esiste sicurezza assoluta e che un sistema informatico, benché protetto con politiche di security elevate, prima o poi sarà vittima di una intrusione. Le proposizioni *“Che cosa stai cercando di*

fare? Perché stai operando in questo modo?” descrivono perfettamente il **modus operandi dell'IDS** che risulta essere complementare a quello visto in precedenza. Seguirà una trattazione più ampia nel capitolo seguente.

2. Il concetto di Intrusion Detection System

Gli Intrusion Detection System svolgono nel campo della sicurezza dell'informazione lo stesso ruolo che possiede l'**antifurto** della nostra auto: quello di individuare per tempo le azioni illecite di possibili ladri.
Questo scopo è raggiunto combinando gli allarmi generati dall'IDS al momento dell'intrusione con l'azione tempestiva del responsabile informativo[1] che provvedere ad attuare le opportune azioni di verifica.
Gli IDS entrano in gioco nella difficile fase di rilevamento dell'intrusione, mentre i sistemi tradizionali affrontano il problema secondo il paradigma risorsa-utente. Analogamente a quanto succede per la nostra auto l'antifurto è solo l'atto finale di una messa in sicurezza della stessa, attraverso l'utilizzo di serrature e chiavi. Allo stesso modo, i sistemi di sicurezza basati sul paradigma C.I.D. rafforzano il perimetro del sistema informatico dalle possibili minacce per mezzo di firewall, proxy e quant'altro e il sistema stesso da attacchi locali mirati ad elevare i privilegi dell'attaccante.[2]

Il **concetto** di Intrusion Detection System è stato coniato nel 1980 da P. Anderson [1] e recita: “Intrusion detection systems analize information about the activity performed in a computer system or network, looking for evidence of malicious behaviours”. Benchè siano passati più di due decenni dalla definizione di Intrusion Detection System e molto è stato scritto da allora in letteratura, solamente negli anni '90 videro la luce le prime implementazioni funzionanti di questi sistemi, sebbene molto più semplici di quelle proposte in teoria.

E. Lundin e E. Jonsson in un loro paper dal titolo *“Survey of Intrusion Detection Research”* [2] propongono un **modello generico** di IDS che è illustrato in figura 1.

[1] L'amministratore o il Site Security Officer (SSO).
[2] La procedura di messa in sicurezza di un sistema informatico è conosciuta con il nome di hardening.

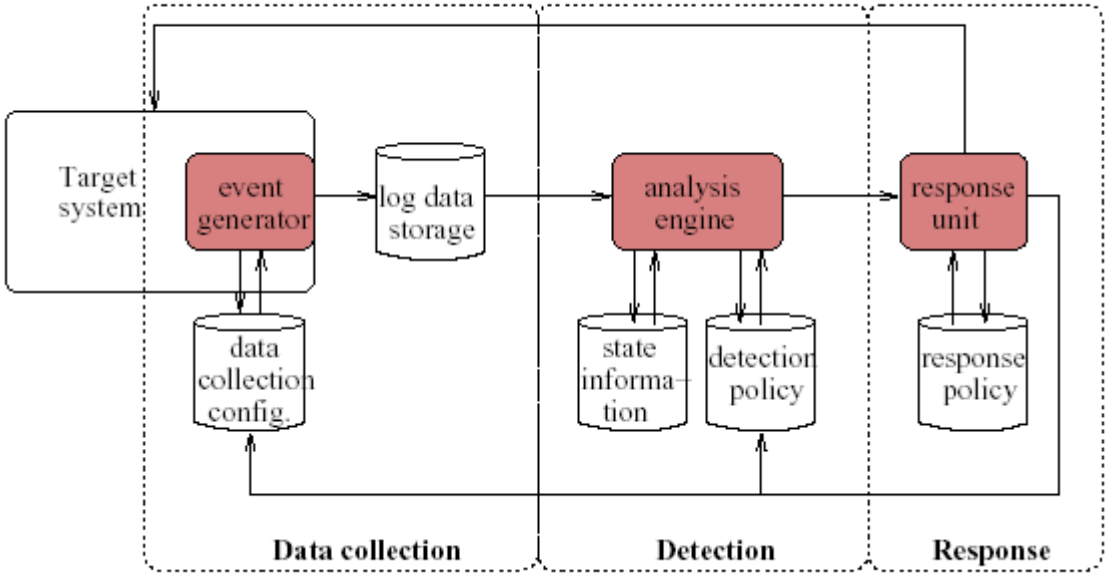


Fig. 1 – L'architettura generica di un Intrusion Detection System

L'*Event Generator* svolge la funzione di data collection: recupera i dati dal sistema monitorato (*target-system)*[1] e li traduce in un linguaggio comprensibile all'Intrusion Detection System a seguito delle direttive definite nella D*ata Collection Configuration*. L'Event Generator può richiedere attivamente i dati al target-system che provvederà ad inviarne una copia o acquisire passivamente le informazioni che quest'ultimo produce (i log per esempio). Conseguentemente al processo di normalizzazione, utile soprattutto per omogeneizzare i dati provenienti da target-system differenti, le informazioni sono passante al componente d'analisi e contemporaneamente vengono depositate in un Log Data Storage mantenendo una History dei dati per analisi future.

Il motore del sistema è l'A*nalysis Engine* che implementa tutti gli algoritmi di rilevazione delle intrusioni. L'Analysis Engine fa fortemente uso del D*etection Policy* in cui son depositate le informazioni preprogrammate su come rilevare le intrusioni e tutte le policy definite dall'amministratore. L'Analysis Engine può essere molto diverso da un IDS all'altro al contrario degli altri componenti dell'IDS che rimangono all'incirca simili. Vedremo nel prossimo capitolo alcune implementazioni di Analysis Engine diverse in funzione delle famiglie di Intrusion Detection.

L'unità finale di R*esponse Unit* risponde in *se*guito di un evento positivo con opportune

[1] Da ora in poi si farà riferimento al sistema monitorato con il termine target-system o soggetto.

azioni. Il comportamento della Response Unit è personalizzato sulla base delle policy inserite nel database di R*esponse Policy*. Un IDS tradizionale, come detto all'inizio del capito, si limita ad avvisare il SSO a cui è relegato l'onere di un controllo approfondito; un IDS distribuito può generare un alert per un secondo IDS; in un **IPS (Intrusion Prevention System)** la Response Unit genera un'azione retroattiva sul target-system o su un componente “esterno” per prevenire una intrusione futura dello stesso tipo o da parte della stessa fonte, oppure per riportare il sistema nelle condizioni di funzionamento “corretto”.

3. Tassonomia degli Intrusion Detection System

In letteratura sono state date numerose classificazioni dei sistemi Intrusion Detection e la più scontata è quella che si basa sull'**approccio** utilizzato nel processo di detection.

3.1 **Classificazione per approccio**

Axelson in "Research In Intrusion-Detection System: A Survey" [3] identifica due grosse categorie che sfruttano due approcci profondamente diversi:

- *Anomaly-detection:* il sistema calcola statisticamente e reagisce a deviazioni "significative" dal comportamento normale del soggetto.[1] "Normale" è definito in relazione al comportamento osservato precedentemente sul soggetto, ed è tipicamente aggiornato quando è disponibile una nuova conoscenza su di esso. Il sistema modella e autoapprende i nuovi profili di comportamento: questo aggiornamento è periodico e automatico. Anomaly detection può essere fatto in molti modi, per esempio attraverso reti neurali [DBS92] e analisi statistica complessa [JV94].
- *Signature-detection (o misuse-detection):* il sistema ricerca l'evidenza di una intrusione all'interno dei dati che corrispondono a firme (signatures) note di un comportamento intrusivo o anomalo. Solitamente queste firme sono costruite off-line, manualmente, man mano che nuovi tipi di attacchi sono resi pubblici alla comunità (per esempio inserendoli nel database Common Vulnerability and Exposures o CVE [5]) e sono mantenute in un database di conoscenza dell'IDS.

Ciascuno di questi approcci ha pregi e difetti che riassumiamo brevemente.
Anomaly-detection:

- Pregi: l'operatore non deve definire manualmente i profili normali di funzionamento del soggetto dal momento che l'IDS è in grado di auto-apprenderli. L'Intrusion Detection System, non possedendo un database di conoscenza definito a priori, è in grado di identificare intrusioni che sfruttano vulnerabilità nuove e non ancora rese pubbliche ("attacchi 0 day") così come variazioni di attacchi noti.
- Difetti: l'approccio anomaly-detection soffre di un alto numero di falsi negativi. Quando la distinzione tra anomalo e normale è poco marcata il sistema può

[1] Per soggetto si intenda un utente, un host o una tipologia di rete.

facilmente classificare come corretto un comportamento che invece non lo è. Un scenario che mette in luce questo problema è quello in cui l'attaccante, cambiando lentamente il proprio comportamento nel tempo, costringe l'IDS ad attribuirgli un profilo di normalità tale da pregiudicarne l'efficienza in fase di detection. Questo problema comporta un alto numero di falsi negativi (vedi poco dopo).

Signature-detection:

- Pregi: l'efficienza di un IDS che sfrutta un approccio di questo tipo è direttamente proporzionale alla qualità delle firme che adotta. Per cui in presenza di un database costantemente aggiornato e gestito con cura si può in linea teorica affermare che il numero di falsi positivi si mantiene basso.
- Difetti: mantenere una base di conoscenza delle firme efficiente (vasta e in continuo aggiornamento) è un'attività difficile e richiede molte energie. Sicuramente questo metodo ha limitate capacità predittive poiché non è in grado di rilevare comportamenti intrusivi per cui non esistano delle firme corrispondenti nel database.

Oltre agli approcci anomaly e misuse detection, un terzo approccio di tipo *Ibrido* integra entrambe le due categorie viste in precedenza. Sebbene gli Intrusion Detecion System ibridi traino vantaggio dai benefici offerti dai due approcci, vi sono tuttavia problemi di metriche decisionali e di falsi positivi che ne limitano il numero di implementazioni.

3.2 **Classificazione per caratteristiche**

Senza entrare troppo nello specifico è facilmente classificare gli Intrusion Detecion System anche in funzione delle principali **caratteristiche** del sistema:

- tempo d'analisi: i sistemi che analizzano il flusso dei dati in tempo reale sono definiti *real-time* o *on-line,* quelli che lavorano a posteriori sui dati catturati si chiamano *non-real-time* o *off-line*. E' ben non confondere un sistema di analisi forense con quello di un IDS offline: il primo è utilizzano a seguito di una intrusione per costruire lo scenario d'attacco ed è eseguito occasionalmente, il secondo è attivato periodicamente e, secondo la definizione di IDS, è utilizzato per rilevare possibili anomalie o intrusioni.
- granularità del processo d'analisi: questa caratteristica permette di suddividere

gli IDS tra quelli che processano i dati *continuamente* e quelli che lo fanno per processi (*batch*). Anche per questa classificazione è bene non confondersi: un sistema off-line può processare i dati continuamente come uno on-line può farlo per piccoli "lotti".

- sorgente dei dati: la/e sorgente/i dati dell'Event Generator (a tal proposito vedere la fig. 1) è una caratteristica talmente importante dell'Intrusion Detection System che ne determina il tipo. Le due maggiori sorgenti sono il flusso dati della rete (*network-data*) e le informazioni locali del target-system (*host-data*)[1]. Seguirà nel paragrafo 3.1 una discussione approfondita dei tipi di Intrusion Detection System in funzione della sorgente dati acquisita.
- risposta del sistema: due categorie di IDS si contendono in questa sezione: *passivi (on-line)* e *attivi* (*in-line*). Sono stati espressi numerosi pareri contrastanti a riguardo di questa classificazione, ed è facile incorrere in discussioni in merito all'argomento nelle mailing-list che trattano la tecnologia degli Intrusion Detection System[2]. E' mia abitudine identificare gli IDS passivi con quelli "tradizionali", che a seguito di una intrusione si limitano ad avvisare le persone preposte al controllo del target-system di quanto accaduto; al contrario gli IDS attivi intervengono attivamente sul target-system modificandone lo stato. Gli IDS attivi, che preferisco chiamare *in-line*, sono la novità di questi ultimi anni, e molti vendor stanno investendo in questa tecnologia che hanno battezzato *Intrusion Prevention System*, il cui acronimo è IPS. Molti di voi mi riterranno tradizionalista dal momento che non credo nella bontà degli IPS. Il maggior motivo che mi ha spinto a considerare gli IPS una strategia commerciale adottata da molti vendor piuttosto che una tecnologia innovativa è la seguente. Il grosso problema di tutti gli IDS è l'alto tasso di errori commessi nel processo di rilevamento delle intrusioni. In letteratura sono stati identificate due grosse categorie di errori: i **"falsi negativi"** rappresentano l'incapacità di un IDS di segnalare un caso in cui un attacco è riuscito a compromettere il target-system; i **"falsi positivi"** rappresentano una segnalazione di attacco o tentato attacco quando non esistono vulnerabilità corrispondenti e l'intrusione non è andata a buon fine. Se gli errori di "falso negativo" consistono in un mancato riconoscimento della forma d'attacco e pongono un limite all'efficacia dell'IDS,

[1] Log del kernel, log degli applicativi, attributi del filesystem, configurazione della memoria, etc..
[2] La più conosciuta è Focus-IDS di SecurityFocus. Trovate un riferimento del capitolo 10.

quelli di “falso positivo” sono ancora più preoccupanti perchè riempiono il SSO di allarmi falsi e sbagliati! Tornando al paragone con l'antifurto della nostra macchina, un falso positivo è quello in cui scatta l'allarme dell'antifurto sebbene nessuno abbia provato a rubarci la macchina. Quante volte ci siamo lanciati in folle corsi nella speranza di trovare il ladro e accorgendosi successivamente di esser stati fregati da un tuono o da una foglia “atterrata” nell'auto? Parlerò più approfonditamente di questo dilemma nel capitolo 5. Se per gli Intrusion Detection System il problema dei “falsi positivi” è grave e comporta falsi allarmi per il SSO, per gli Intrusion Prevention System lo è ancor di più. Quando l'Analysis Engine dell'IPS sbaglia e commette un errore di “falso positivo”, interviene con una retroazione errata sul target-system che, nelle migliori delle situazioni, corrisponde a un blocco di traffico legittimo, cosa ben più grave! Concludendo, a mio avviso costruire un nuovo edificio su un terreno poco solido, è come porre le basi di un nuovo sistema software sull'instabilità di quello esistente, ignorando i problemi che lo caratterizzano: primo tra tutti l'eccessivo numero di “falsi positivi”. Credo che prima di estendere il concetto di IDS con quello di IPS sia necessario garantire alla struttura software originaria una stabilità e robustezza tale da permetterne un'evoluzione sicura.

3.3 **Network-based e Host-based IDS**

All'interno del capitolo sulla tassonomia degli IDS è d'obbligo dedicare un paragrafo alle due grosse famiglie di Intrusion Detection System: quelli *network-based* (*NIDS)* e quelli *host-based (HIDS)*.
Come anticipato nello scorso paragrafo il tipo di dato acquisito dal sistema anti-intrusione è di fondamentale importanza, tale da influenzarne la famiglia d’appartenenza: il sensore di un network-based IDS gestisce una sorgente “network-data”, quello di un host-based IDS una sorgente “host-data”.

Dal momento che la classificazione “per approccio” (anomaly vs misuse) è ortogonale rispetto a quella “per sorgente dati” (host vs network), un network-based IDS può adottare sia un approccio anomaly-detection che misuse-detection (anche entrambi). Tuttavia è più facile trovare funzioni di misuse-detection all'interno di un NIDS e, viceversa, funzioni di anomaly-detection in un HIDS.

Storicamente gli **host-based IDS** furono i primi IDS a venire sviluppati e basano il loro funzionamento sul controllo “in loco” di log, syscall, file e qualsiasi altra funzione/variabile caratteristica del target-system, al fine di individuare possibili azione illecite nei confronti del sistema stesso. Dal momento che gli IDS host-based controllano un singolo soggetto (un host o un'applicazione), ne esistono di molti tipi a seconda del sistema operativo e del software installato (vedi paragrafo 4.1).

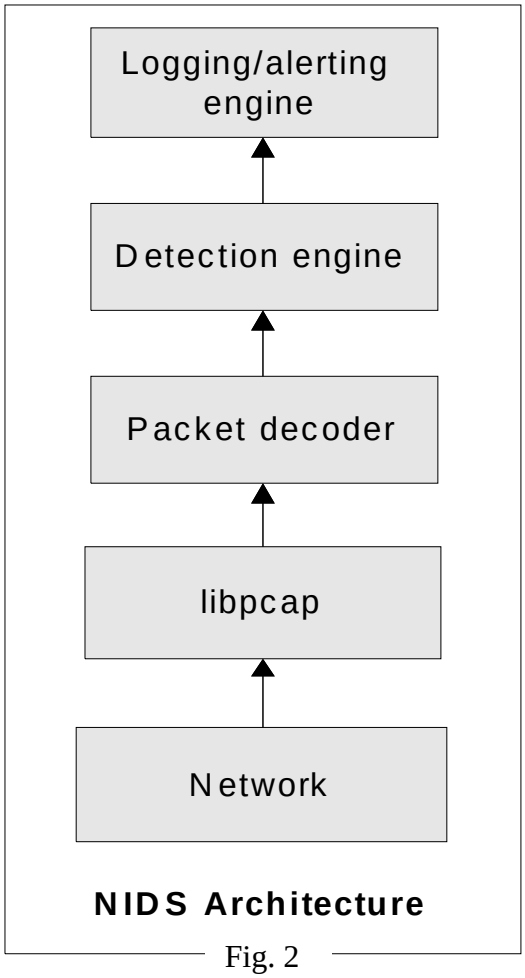


Fig. 2

Un **network-based IDS** presenta un'architettura a layer: il sensore di cattura il traffico di rete, lo invia all'unità d'analisi che svolge opportune funzione di decodifica, normalizzazione e analisi.

La procedura di lettura in real-time del traffico è spesso implementata attraverso una libreria di rete di basso livello[1]. Il sensore deve essere collocato opportunamente all'interno del sistema distribuito in modo da poter catturare tutte le informazioni necessarie per poter operare. Questa necessità spesso si traduce in complicate acrobazie da parte dell'installatore del sistema di intrusion detection. Con la veloce diffusione degli apparati di switching tutte le reti sono logicamente frammentate e il sensore deve essere collegato alla porta di SPAN dello switch dopo averlo opportunamente configurato per replicare tutto il traffico verso il sensore [CISCO1]. Una seconda possibilità è inserire tap passivi nei segmenti che l'IDS deve monitorare. L'esperienza vuole che statisticamente molti tap abbiano grossi problemi nel loro funzionamento. Questi problemi non si presentavano per le “vecchie“ reti hubbate in cui era sufficiente collocare il sensore in una porta qualsiasi dell'hub.

[1] La più conosciuta e utilizzata è la Libpcap [4] distribuita con licenza Opensource BSD [BSD].

Questo ed altri limiti dei NIDS saranno discussi nel capitolo dedicato.

In un network-based IDS con approccio misuse-detection l'attività di analisi svolta dall'Analysis Engine è basata su un **meccanismo di pattern matching** tra il traffico di rete e la base dati delle firme. Nell'evoluzione di ogni Intrusion Detection System gli algoritmi di pattern matching sono diventati più sofisticati, complessi e inevitabilmente più onerosi in termini di risorse di sistema. In letteratura sono stati proposte molte soluzioni, alcune delle quali sono state implementate da vendor più o meno famosi. A tal proposito il paper di Sourcefire *Snort™ 2.0 - Detection Revisited* [SF1] è un buon punto di partenza nell'approfondimento del funzionamento di un Analysis Engine per un IDS network-based con approccio signature-detection. Atri documenti degni di nota sono [SF2], [SF3] e [SF4]. Piccola nota a margine, Sourcefire è il vendor che ha contribuito maggiormente al finanziamento e allo sviluppo del più famoso NIDS Opensource: Snort[1] [SNORT].

Così come assieme al bianco e al nero esistono sempre delle sfumature grigie intermedie, tutte le suite IDS più popolari utilizzano un approccio misto ottenendo un sistema anti-intrusione capace di controllare l'intera rete e le singole macchine allo stesso tempo. Una soluzione ibrida sfrutta le caratteristiche positive di entrambe le tipologie di Intrusion Detection, ma allo stesso tempo deve far fronte a un numero di falsi positivi ancora superiore.

[1] Per maggiori informazioni si faccia riferimento al paragrafo 4.2.1

4. Sistemi ed architetture rilevanti

L'intento di questo capitolo è di presentare le implementazioni più popolari di Intrusion Detection System (con l'eccezione di snort_inline [SI] che è il corrispondente IPS di Snort).
Ritengo più istruttivo e coerente con l'approccio di ricerca presentare software Opensource dal momento che tale licenza da diritto al suo utilizzatore di provare, studiare, modificare e distribuire il software stesso senza alcuna forma di copyright[1]. Nella bibliografia ho riportato i link da cui prelevare gratuitamente e sotto forma di codice sorgente tali applicativi.

4.1 **Host-based IDS**

Esistono tre grosse famiglie di HIDS:

- Filesystem-monitoring: controllano il target-system attraverso un confronto periodico tra un DB "trusted" e il filesystem attuale (firma, data e dimensione dei file). Alcuni esempi sono TripWire [TW] e Aide [AI].
- Log-monitoring: controllano l'attività del sistema per mezzo dell'analisi dei log del kernel o di particolari processi. (Swatch [SW], Logsurfer [LS], Logwatch [LW]).
- OS-monitoring: controllano le attività di base del sistema operativo direttamente in kernel space, per esempio attraverso l'uso di moduli del kernel (Grsecurity [GR], KSTAT [KSTAT]).

Come sempre, esistono HIDS appartenenti ad una singola famiglia e numerosi altri ibridi.

4.1.1 Tripwire

Tripwire è un HIDS di tipo "filesystem monitoring". Il suo funzionamento è abbastanza semplice (vedere la figura 3). La prima fase è quella di installare Tripwire sul target-system e di creare il DB "trusted" prima che il sistema venga messo in produzione (reso pienamente operativo) (punto 1): questa procedura ci garantisce che tutte le informazioni raccolte nella base dati siano corrette. Durante il normale funzionamento

[1] La licenza Opensource più utilizzata è la GPL [GPL].

(punto 2), Tripwire controlla quegli attributi dei file che non dovrebbero modificarsi, come la loro signature (ottenuta mediante hashing MD5), la loro dimensione, la data di creazione, etc. Quando si verifica una discordanza con la base dati costruita nel punto 1 lo strumento genera un allarme per l'amministratore (punto 3).

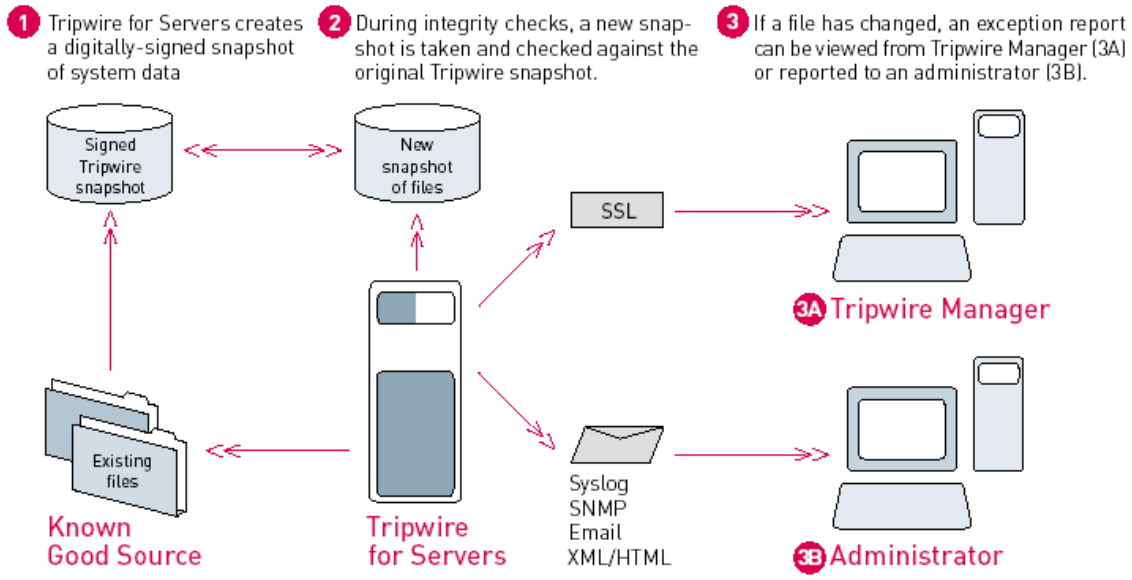


Figura 3. L'architettura di Tripwire

In questo modo, qualsiasi modifica dell'attaccante al filesystem (precisamente solo a quella parte di filesystem controllata da Tripwire) è tempestivamente segnalata al SSO.
Ai più attenti tra voi sarà sorta una domanda spontanea: *"quali file far controllare a Tripwire?"*. La risposta a questo quesito sta nella risposta a un secondo quesito: *"quali file si presume andrà a modificare il nostro attaccante?"*. Ritengo sia molto importante per un site security officer conoscere il comportamento tipico dei suoi avversari. "Far sicurezza" è come giocare una partita con il nostro attaccante: sarà persa in partenza qualora l'attaccante abbia uno skill[1] troppo elevato rispetto a quello dell'amministratore.
Una volta ottenuto l'accesso ad un sistema, l'interesse maggiore dell'attaccante diventa quello di mantenerlo per un tempo più lungo possibile. Questa necessità corrisponde a quella di nascondersi accuratamente all'occhio dell'amministratore che supervisiona le

[1] Termine frequentemente utilizzato in ambito informatico per indicare l'abilità di una persona.

attività dello stesso. Esistono molte tecniche, più o meno raffinate, per assolvere questo compito: quella utilizzata dipenderà dallo skill dell'attaccante. Molte di queste richiedono la modifica di particolari files: i log di un demone, le configurazioni di un programma, i binari e le librerie di sistema. E' in queste situazioni che un HIDS come Tripwire genera un allarme avvisando l'amministratore di un comportamento anomalo causato, per l'appunto, dalla presenza di un intruso.

4.1.2 GrSecurity

Lo stesso effetto viene raggiungo da GrSecurity sfruttando un approccio complementare: il controllo del sistema in kernel space. Attraverso opportune modifiche alle chiamate di sistema e ai meccanismi di gestione delle risorse, GrSecurity è in grado di riconoscere tempestivamente numerosi attacchi portati dall'attaccante per percorrere la scalata di privilegi verso root o per provocare un DoS.
GrSecurity è allo stesso tempo un tool di hardening e un host-based IDS con le seguenti caratteristiche:

- Protezione contro i più comuni metodi per exploitare un sistema:
  - Modifica dello spazio d'indirizzamento
  - Race condition (specialmente in /tmp)
  - Rottura di un ambiente chrottato
- Ricco sistema di ACL con un tool di amministrazione user-space
- Supporto per sysctl (modifica al volo via procfs)
- Meccanismo di logging degli attacchi e auditing di alcuni eventi:
  - Exec()
  - Chdir(2)
  - mount(2)/unmount(2)
  - Creazione ed elimina di IPC (semafori, code di messaggi)
  - Fork fallite
- Supporto per l'architettura multi-processore

4.1.3 KSTAT

Completo la panoramica degli host-based IDS in kernel space citando il tool di Matteo

Falsetti (FuSyS): KSTAT[1].
KSTAT assiste il SSO nella ricerca e nella rimozione di moduli troiani (LKM) o rootkit, spesso utilizzati dall'attaccante per nascondere la propria attività all'interno del sistema.
L'utilizzo di un rootkit in kernel space influenza di riflesso tutte le applicazioni user space che fan richiesta di un particolare servizio, per esempio la modifica della lista circolare in cui il S.O. mantiene i processi "attivi" impedisce a programmi come ps e top di vederne la presenza. In questo modo un attaccante può facilmente nascondere all'occhio dell'amministratore un qualsiasi codice in esecuzione: uno sniffer piuttosto che un keylogger, per esempio. Uno dei più famosi rootkit è Adore del gruppo hacker tedesco Teso [AT].
Giunto oramai ad una versione matura, KSTAT supporta il dump dei socket di rete, il fingerprinting delle chiamate di sistema, lo scanning dei moduli nascosti ed altro**.**

4.2 **Network-based IDS**

Per quanto riguarda i network-based IDS non sono mai state individuate delle vere e proprie famiglie di appartenenza, piuttosto esistono NIDS che sfruttano l'approccio anomaly based ed altri, la maggior parte, che utilizzano un approccio signature-detection.

4.2.1 Snort

Snort è un network-based IDS real-time, lightweight, con risposta passiva alle intrusioni e con un'ottima accuratezza nel rilevamento delle intrusioni che sfruttano vulnerabilità note. Snort possiede caratteristiche realtime: è in perenne ascolto sul segmento di rete a cui è stato adibito ed è in grado di contattare tempestivamente il SSO attraverso email, syslog e i messaggi Winpopup. Non è in grado di bloccare i pacchetti considerati maliziosi, ma si limita a informare l'amministratore con i meccanismi sopra elencati.
La leggerezza di questo software deriva principalmente da due caratteristiche, la sua ridottissima dimensione (i sorgenti di Snort hanno una dimensione inferiore ai 600KB) e l'efficienza nella scansione dei pacchetti che lo rende utilizzabile anche su hardware non molto performante.
L'IDS in questione adotta un approccio di signature-detection attraverso regole che

[1] KSTAT è utilizzato di frequente anche nelle attività di analisi forense.

possono essere molto specifiche e ispezionare l'intero campo dati del pacchetto di rete. Questo consente la scrittura e la personalizzazione di regole che identificano univocamente pacchetti generati per sfruttare attacchi note come i buffer overflow [Ale96] o i cgi-probe [CGI]. La semplicità di scrittura delle regole e il basso layer di funzionamento lo hanno reso uno strumento particolarmente duttile [Roe99], anche come strumento di analisi di attacchi sconosciuti.

Altre informazioni sono reperibili dal sito internet di Snort [SNORT] e nella FAQ che trovate al seguente indirizzo internet http://www.snort.org/docs/FAQ.txt

4.3 **snort_inline, un esempio di IPS**

snort_inline [SI] è fondamentalmente una versione modificata di Snort che accetta pacchetti da iptables[1] [IPT], attraverso libipq, invece che da libpcap [4]. snort_inline utilizza un nuovo set di regole (drop, sdrop, reject) per indicare a iptables se il pacchetto deve essere scartato, rifiutato, modificato o permesso di passare in accordo alle regole definite in Snort.

[1] Iptables è il programma di firewalling ufficiale di Linux. Iptables si appoggia alla corrispondente parte in kernel space (Netfilter) in linux v. 2.4 e 2.6.

5. I limiti degli attuali Intrusion Detection Sytem

In letteratura sono stati mostrati più volte i limiti degli attuali sistemi Intrusion Detection e di pari passo è stata proposta una lunga serie di soluzioni risolutive. Tuttavia quasi sempre le proposte mosse dalla comunità scientifica di ricerca nel campo degli IDS non hanno trovato il favore dalle società che sviluppano questa tecnologia, che preferiscono rimanere legate alla "tradizione" piuttosto che investire in soluzioni innovative.
In questo capitolo presenterò alcune delle problematiche più evidenti degli Intrusion Detection System tradizionali; queste ed altre carenze hanno motivato le ricerche di un nuovo modello di IDS più efficiente, che verrà discusso nel capitolo 7.

Come già anticipato, esiste un grosso problema che affligge tutti gli Intrusion Detection System attuali: la presenza di falsi negativi e **l'alto numero di falsi positivi** che impedisce al SSO di discernere tra allarmi fasulli e allarmi veramente reali. Da parte di un attaccante è facile sfruttare il problema dei falsi positivi per offuscare la propria traccia di attacco: è sufficiente forzare l'Intrusion Detection System a generare allarmi fasulli, per esempio scrivendo del "rumore" nei file di log controllati dal HIDS (quelli di syslog per esempio) o creando un flusso dati particolare verso la sonda del NIDS.
L'alto tasso di falsi positivi e di falsi negativi è una conseguenza implicita dell'approccio adottato dagli IDS tradizionali: troppo generico e incapace di valorizzare le caratteristiche proprie del sistema controllato. Se da un lato l'utilizzo di procedure generiche permette di utilizzare lo stesso software in contesti di rete differenti e quindi vede il favore dei responsabili commerciali attenti ad ottenere utili maggiori a scapito della qualità, dall'altro diminuisce l'efficienza dell'IDS a scapito della sicurezza dell'intero sistema informativo.

La differenza tra un attaccante e un software che gioca in difesa, come un IDS, è che il primo possiede delle informazioni vaghe, imprecise ed incomplete sulla propria vittima, recuperate con difficoltà tramite una procedura che prende il nome di information-gathering, al contrario il difensore conosce alla perfezione il "sistema vittima"[1], e gioca in vantaggio rispetto al nemico.
L'efficienza di un Intrusion Detection System è proporzionale alla **qualità della**

[1] In riferimento ad un IDS, quello che è stato chiamato Target-System. Vedi paragrafo 2.

sorgente dati[1] su cui si basa il motore d'analisi per ricerca di tracce di intrusioni e di anomalie. Riprendendo la metafora della partita, paradossalmente gli IDS tradizionali non sfruttano i vantaggi offerti dal giocare in difesa, ma utilizzano sorgenti dati povere di informazioni, semplici, elementari, generali e in parte ignorate dall'IDS stesso.
Un lampante esempio di IDS che non valorizza le particolari caratteristiche del target-system monitorato è Cisco IOS® Intrusion Detection System, un modulo per il sistema operativo dei router Cisco che implementa un IDS per il router stesso. Secondo quanto dice Cisco [CISCO2] IOS IDS è uno strumento che realizza una protezione completa e una soluzione sicura per combattere le intrusioni non autorizzate. Cisco IOS IDS appartiene alla famiglia del network-based IDS e utilizza un approccio di tipo misuse-detection. Il suo funzionamento è molto semplice: il traffico entrante ed uscente dalle interfacce del router è confrontato con un knowledge database di 59 firme divise in due categorie: information signature e attack signatures. Le prime cercando di intercettare le attività di information gathering come port scanner ed echo request, le seconde mirano a rilevare attacchi quali DoS o exploit.
Il primo limite riguarda l'impossibilità di estendere e personalizzare il database delle firme, nonostante il numero ridotto di signature. Questa limitazione è abbastanza grave in una realtà come quella attuale in cui le forme d'attacco aumentano quotidianamente.
I più intuitivi tra voi si saranno domandati che reale necessità ci sia di implementare un network-based IDS all'interno di un router. Sarebbe più efficiente inserire le sonde NIDS nei segmenti di rete delle interfacce del router, realizzando una struttura più performante, sicura e meno intrusiva. L'idea di creare un IDS “plasmato” per il router stesso è sicuramente migliore! Infatti, Cisco IOS IDS, essendo un NIDS, non implementa nessun controllo delle funzionalità e delle variabili caratteristiche di un router, al contrario il modello di Intrusion Detection System da me proposto.

Esistono inoltre altri di limiti ben conosciuti dei network-based IDS che sono tra le cause principali dell'alto numero di falsi negativi.
Su una rete molto trafficata **il grosso carico di traffico sovraccarica i sensori** NIDS ed impedisce l’individuazione dei pacchetti riconducibili ad un attacco informatico, proprio come il detto “trovare l’ago nel pagliaio”. La corrispondenza tra migliaia di pacchetti e centinaia di firme è così dispendioso in termini di risorse che è statisticamente provato

[1] A seconda della famiglia di Intrusion Detection System la sorgente dati può essere i log di un'applicazione, i log del kernel, gli eventi del sistema operativo, il traffico di rete, etc..

che un NIDS è in grado di monitorare solamente il 60-80% di un segmento a 100MBps e 40-60% di un segmento Gigabit. Per esempio se supponiamo di monitorare 10 macchine su uno switch a 10Mbps, la porta di SPAN lavora a pieno carico (a 100MBps). Statisticamente un buon 30% del traffico è perso, ciò vuol dire che un terzo degli host è “invisibile” al NIDS e un attaccante può facilmente portare un attacco senza essere identificato.

Alla presenza di **traffico crittografato**, come una transazione https, che usa SSL come algoritmo di crittografia, tutti gli sforzi del NIDS sono vani. Infatti, a meno di conoscere le chiavi con cui sono cifrati i dati, pratica non realizzabile per l'alto costo computazionale e per i problemi di decrypting asimmetrico, uno sniffer di rete come il sensore del NIDS si trova ad affrontare un pacchetto il cui contenuto è crittografato. Le uniche informazioni che può sfruttare sono quelle in chiaro: le intestazioni dei pacchetti (gli header tcp e ip per esempio) nel caso di un protocollo di crittografia a livello d’applicazione come SSL, o il network layer nel caso di protocolli crittografici quali IpSEC.

Infine in una rete a **configurazione asimmetrica** dei router, soluzione utilizzata di frequente nelle realtà enterprise dove deve essere sempre garantito il principio di disponibilità, un NIDS installato sul singolo router non ha la garanzia di catturare stream interi di dati.
Nelle reti a commutazione di pacchetti, a differenza di quelle a circuito, il tragitto seguito da ciascun pacchetto può variare per una serie di motivi non predicibili a priori. Questo è ancor più vero in una soluzione con più entry point alla rete, in ridondanza tra di loro, la cui funzione è proprio quella di distribuire il carico di rete tra i vari router.

Questi ed altri problemi sono stati già stati discussi in letteratura e non mancano delle proposte di **possibili soluzioni**, tra cui quella di bilanciare i flussi dati tra più NIDS per mezzo di un IDS Balancer [TL].

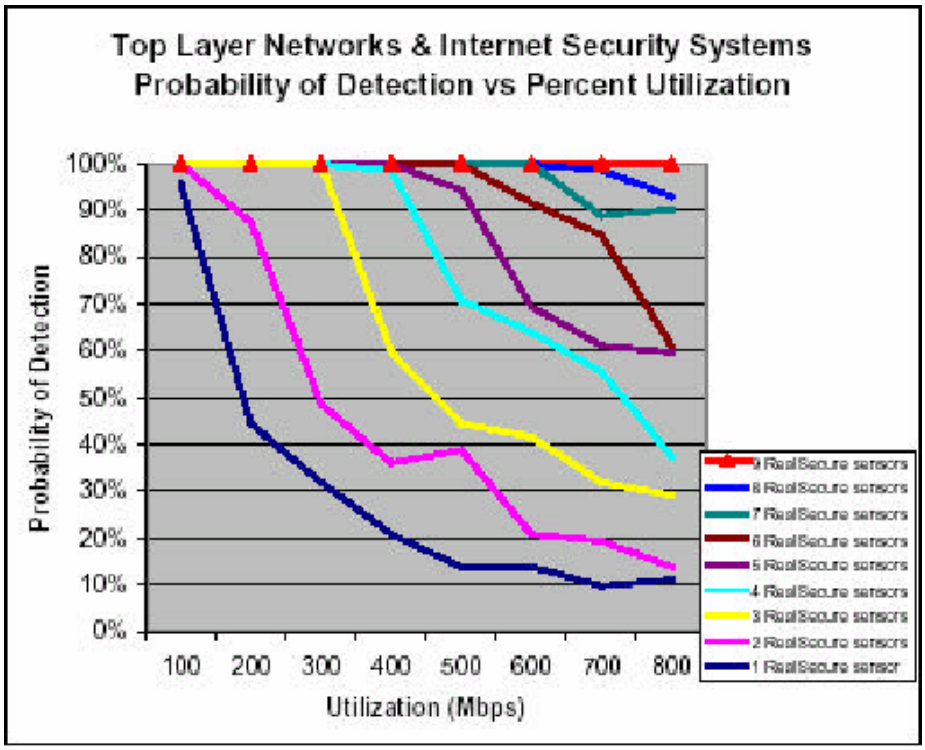


Figura 4. Bilanciamento del flusso dati di un NIDS

Come evidenziano i risultati delle prove effettuate (fig. 4), una batteria di nove NIDS è capace di individuare attacchi noti con una precisione che si avvicina al 100%. Questo risultato sta ad indicare un numero di falsi negativi quasi nullo, tuttavia l'utilizzo di altrettanti IDS comporta un aumento proporzionale di fasi positivi. Inoltre le ripercussioni in termini di costi sono talmente alte che rendono questa soluzione proibitiva per aziende di dimensioni medie e piccole.

Il modello proposto[1] è complementare a quello appena visto: partendo dal presupposto che è troppo ambizioso controllare le attività dell'intera rete per mezzo di una batteria di NIDS, il nuovo modello distribuisce il controllo sugli elementi di maggior interesse del sistema informativo utilizzando un'**architettura distribuita** e a **layer**. Questa soluzione risolve anche un altro forte limite degli HIDS: gli host-based IDS possiedono un **contesto espressivo ristretto al singolo host** controllato[2] e non sono in grado di sfruttare la conoscenza sviluppata dagli altri HIDS del sistema informativo con uno spreco evidente di risorse e di efficienza.

[1] Si veda i prossimi capitoli.
[2] Il target-system.

Facendo un esempio, in una architettura a due server A e B (e due rispettivi HIDS), i risultati delle elaborazioni ottenuti dal primo HIDS non possono essere applicati direttamente dal secondo HIDS, dal momento che i due Intrusion Detection System non possiedono un canale di comunicazione privato per scambiarsi le informazioni. Uno scenario d'attacco questa incapacità è questo: un attaccante ottiene l'accesso al server A e, grazie ad un exploit locale riesce a provocare un Deny of Service. L'Intrusion Detection System utilizza un approccio anomaly-based e, grazie ad una procedura d'autoapprendimento, all'n-esima volta che si verifica l'attacco lo identifica correttamente e avvisa l'amministratore. Se in seguito l'attaccante prendesse possesso del secondo server e realizzasse lo stesso attacco, l'HIDS di B dovrebbe attendere n tentativi prima di riconoscere una connessione tra exploit e DoS. Il modello proposto utilizza una procedura di sincronizzazione tra le basi di conoscenza sviluppate dai singoli Intrusion Detection System; in questo modo il secondo HIDS individua correttamente l'attacco alla prima occasione, dal momento che l'HIDS A ha precedentemente sviluppato una sua conoscenza in merito a quel tipo d'attacco.

6. L'approccio context-based

L'approccio context-based e si basa su tre principi fondamentali: il focus, la conoscenza a priori e la correlazione.

Focus

L'Intrusion Detection System deve focalizzarsi esclusivamente sul controllo dei **comportamenti specifici del target-system**. Questo significa che un router con una specifica versione di software deve ospitare un IDS adattato a processare specifiche firme di attacco e comportamenti anomali. A condizione che l'IDS si focalizzi sugli allarmi significativi di un componente specifico per hardware e software (sia il sistema operativo che le applicazioni), l'approccio context-based offre la più accurata e aggressiva riduzione di allarmi così come la più efficace individuazione di attacchi.

Contemporaneamente una focus più ampia sposta il campo di interesse dell'IDS verso una prospettiva di **comunità** (soggetti omogenei) o di scenario (soggetti eterogenei) del sistema informativo. Per esempio, ci si aspetta che router appartenenti alla comunità di gateway perimetrali di una VPN operino nell'insieme come sistema unitario per supportare i servizi di comunicazione dell'infrastruttura: se consideriamo ciascun router della comunità come un router stand-alone con una applicazione generica, trascuriamo le informazioni utili per affrontare gli attacchi di sicurezza

Quindi, un motore di IDS context-based basato sul concetto di comunità, dovrebbe mettere a fuoco gli allarmi che provengono dalla comprensione degli eventi nella prospettiva del comportamento aggregato.

Parallelamente uno **scenario** è un'aggregazione di componenti informativi eterogenei che assolvono ad un compito comune. Uno scenario potrebbe essere un'applicazione Web con un layer di presentazione esposto ad internet, una logica applicativa in DMZ e un network-based IDS per il controllo del traffico.

Conoscenza a priori

Come è già stato anticipato nel capitolo dedicato ai limiti degli IDS tradizionali, uno dei principi più importanti nel fare sicurezza è quello per cui i difensori non devono ignorare le informazioni disponibili del sistema da difendere. Una **base dati aggiornata** costantemente con le **informazioni del target-system** è una condizione obbligatoria per assicurare un filtering semi-automatico dei falsi positivi.

L'approccio più brutale è applicato dai network-based IDS tradizionali che pretendono di individuare le attività maligne catturando e utilizzando sequenze di byte come firme di confronto, ma ignorando il contesto in cui sono collocati a funzionare (approccio cieco).

In contrasto, l'approccio lo context-based si basa sul principio della conoscenza a priori, proposto già da alcuni ricercatori e vendor. La **personalizzazione precisa della configurazione** dell'Intrusion Detection System in accordo al contesto operativo dell'IDS, ai risultati delle sessioni di ricerca delle vulnerabilità e la selezione degli allarmi sulla base dei possibili rischi del sistema e delle vulnerabilità non corrette, riduce drasticamente il numero di falsi positivi garantendo allo stesso tempo massima efficienza nell'individuazione degli attacchi.

L'applicazione del principio della conoscenza a priori a un network-based IDS porta con se risultati straordinari anche in scenari di utilizzo molto semplici, faccio un esempio: l'attaccante prova un attacco con un exploit per il servizio ftp di linux (proftpd per esempio) ma non ottiene l'accesso perché il server vittima usa un sistema operativo Windows. In questa situazione un NIDS tradizionale genera un falso positivo perché la corrispondenza pacchetto-firma è corretta e il sistema non ha altre informazioni da utilizzare nella valutazione; invece un IDS context-based possiede una base di conoscenza più completa che include, tra le tante cose, il sistema operativo delle macchine monitorate, che gli permette di scartare gli eventi di attacchi non riferiti al sistema operativo e ai servizi offerti dal target-system.

Correlazione

Per affrontare le limitazioni di informazioni utili fornite come output dagli IDS tradizionali, è stato previsto l'uso di meccanismi di correlazione per **arricchire il contenuto informativo** e per aggiungere un ulteriore livello di astrazione capace allo stesso tempo di **priorizzare** e **contestualizzare** gli allarmi quanto tenere in considerazione gli eventi grezzi[1] forniti a basso livello dagli IDS. La correlazione degli allarmi è basata sull'**analisi congiunta degli eventi provenienti da più IDS** in una finestra temporale comune o secondo una prospettiva storica. Questo approccio raccoglie un ampio numero di eventi grezzi e, attraverso vari processi di filtering (eventi che non portano ulteriore informazione sono eliminati) e clustering (più eventi grezzi sono combinati in un singolo evento più significativo), sintetizza ricche informazioni

[1] Viene definito grezzo l'evento generato dal singolo IDS.

sulla rilevazione delle intrusioni e permette di ridurre il numero di falsi positivi/negativi e di attacchi non rilevanti al contesto del target-system.

Questi principi suggeriscono un modello di Intrusion Detection System che si adatti al contesto di operatività dello stesso. E' naturale **modificare la tassonomia host-based vs network-based** presentata nel paragrafo 3.3 aggiungendo altre famiglie di Intrusion Detection System, specifiche per sistema operativo, per applicazione, per servizi offerti, per funzionalità (client, server), per tipologia di apparato (tipo, marca e modello), per contesto di rete (intranet, DMZ, testing, honeypot), per aggregazione di dispositivi (comunità, scenario) e quant'altro[1].

Il resto del documento è focalizzato sull'analisi del modello **Router-IDS**[2] da me sviluppato seguendo i principi dell'approccio context-based. RIDS [RIDS] è un Router-IDS realizzato esclusivamente come **esempio** concreto di IDS context-based e a fini strettamente didattici; non deve essere utilizzato in produzione e copre solamente alcune (poche!) delle problematiche relative alla sicurezza di un router; il framework di supporto per la correlazione di eventi a contesto di comunità/scenario non è stato volontariamente sviluppato.

[1] Per esempio ci può essere un IDS per il Router Cisco (Router-IDS), uno per il Firewall Checkpoint (Firewall-IDS) e uno per il Mailserver Linux (Mailserver-IDS).
[2] Router-IDS è la famiglia di IDS di cui fa parte. Il nome del programma è RIDS.

7. Il modello Router-IDS e una sua implementazione: RIDS

Il Router-IDS è un innovativo modello di Intrusion Detection basato sui tre principi dell'approccio context-based: focus, conoscenza a priori e correlazione. Il modello proposto estende la tassonomia degli host-based/network-based con una famiglia di IDS per routers. RIDS è un progetto Opensource di Router-IDS che utilizza un approccio ibrido (anomaly e misuse-detection) per il controllo delle anomalie e delle intrusioni dei routers. L'architettura di RIDS rispecchia il modello di Router-IDS ed è descritta nel paragrafo 7.2.

I **goal** di questo progetto sono:

- alte prestazioni: IDS embedded nel router (come Cisco IOS IDS) appesantisco il processore dell'apparato peggiorandone le prestazioni e aumentando il rischio di fault hardware. Uno strumento remoto garantisce prestazioni migliori;
- indipendenza: è possibile aggiornare il knowledge database di RIDS, aggiungere funzionalità e modificarne il comportamento senza intervenire sul router;
- bassa intrusività: RIDS è facilmente integrabile con altre soluzioni security-oriented andando a garantire più efficacemente il paradigma C.I.D.[1];
- flessibilità: utilizzando focus specifici per marca, modello e funzionalità del router, RIDS massimizza la conoscenza del target-system con tutti i vantaggi che ne derivano (vedi capitolo precedente);
- maggior sicurezza: una volta che il router è stato violato l'attaccante possiede l'autorità per modificarne la configurazione[2]. E' in grado di disattivare le funzioni di controllo e di nascondersi all'occhio dell'amministratore di sistema. Un IDS remoto, almeno nelle prime fasi dell'attacco (quella di information gathering e del primo accesso), registra in modo irreversibile gli eventi (per esempio su un supporto a sola lettura). Successivamente l'attaccante sarà sempre in grado di occultarsi modificando la configurazione del router, ma la traccia dell'avvenuto attacco rimarrà registrata in remoto;
- basso numero di falsi positivi/negativi: utilizzando l'approccio context-based, RIDS genera un basso numero di falsi positivi e negativi;
- conoscenza distribuita: attraverso una architettura distribuita (si veda paragrafo

[1] Vedi capitolo inziali.
[2] Il sistema operativo di un router Cisco si chiama Cisco IOS e la configurazione corrente è disponibile nel file "running-config" o con il comando #show running-config.

7.2) la base dati di conoscenza di un IDS è condivisa dall'intera struttura di rilevamento delle intrusioni, superando la limitazione di contesto degli host-based IDS illustrata al termine del capitolo 5.

Il prossimo paragrafo descrive il protocollo SNMP, utilizzato nell'interfacciamento con il target-system per la sua semplicità e portabilità.

7.1 **Simple Network Managment Protocol (SNMP)**

SNMP [SNMP] è un protocollo applicativo molto diffuso per il controllo remoto di molteplici dispositivi di rete come stampanti, router, access point, switch e server. Tutte le versioni di Cisco IOS [IOS] supportano SNMP.

Attualmente esistono tre versioni di SNMP: SNMPv1 è stata la prima implementazione del protocollo SNMP, sviluppata nel 1990 e descritta dall'RFC 1157. Data la struttura troppo semplice e limitativa della prima versione, ci sono stati vari tentativi di rimpiazzarla con una successiva: nel 1996, secondo l'RFC 1907, nasce SNMPv2, la cui sottoversione SNMPv2C (Community-based Administrative Framework, RFC 1901) è lo standard de facto nel mondo odierno. SNMPv2c aggiunge molte novità al suo predecessore, tra cui: le PDU di GetBulk e Inform, una estensione dei messaggi di errore e una indipendenza maggiore dal protocollo di trasporto. La terza versione del protocollo nasce, in special modo, per sopperire alle mancanze dei suoi predecessori nell'ambito della sicurezza delle trasmissioni. Le politiche di sicurezza sono implementate tramite crittografia, tabelle hash e altri strumenti che consentono l'autenticazione dei pacchetti, delle password e, anche, delle PDU. Tramite diversi livelli di sicurezza si può stabilire se consentire un accesso senza autenticazione (no pwd/no priv), con autenticazione (pwd/no priv) o con autenticazione e codifica dei dati (pwd/priv).

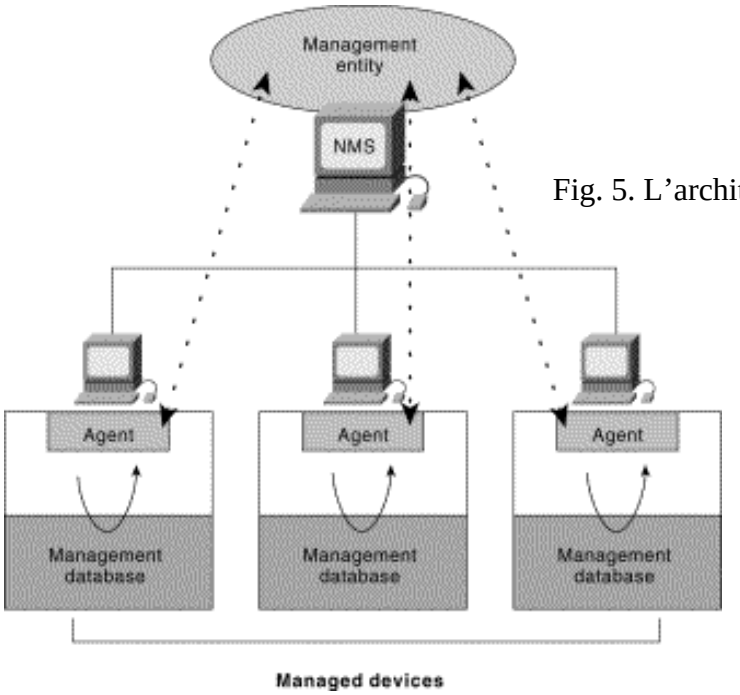


Fig. 5. L'architettura SNMP

Un sistema di controllo SNMP è composto da tre componenti: i dispositivi monitorati (managed devices), l'agente (un software che sta sulla piattaforma monitorata) e il manager che prende il nome di NMS (network-management system).
Solitamente le trasmissioni SNMP utilizzano il protocollo UDP (porte 161 e 162) e la community come credenziale di autenticazione (almeno per SNMPv1 e SNMPv2).

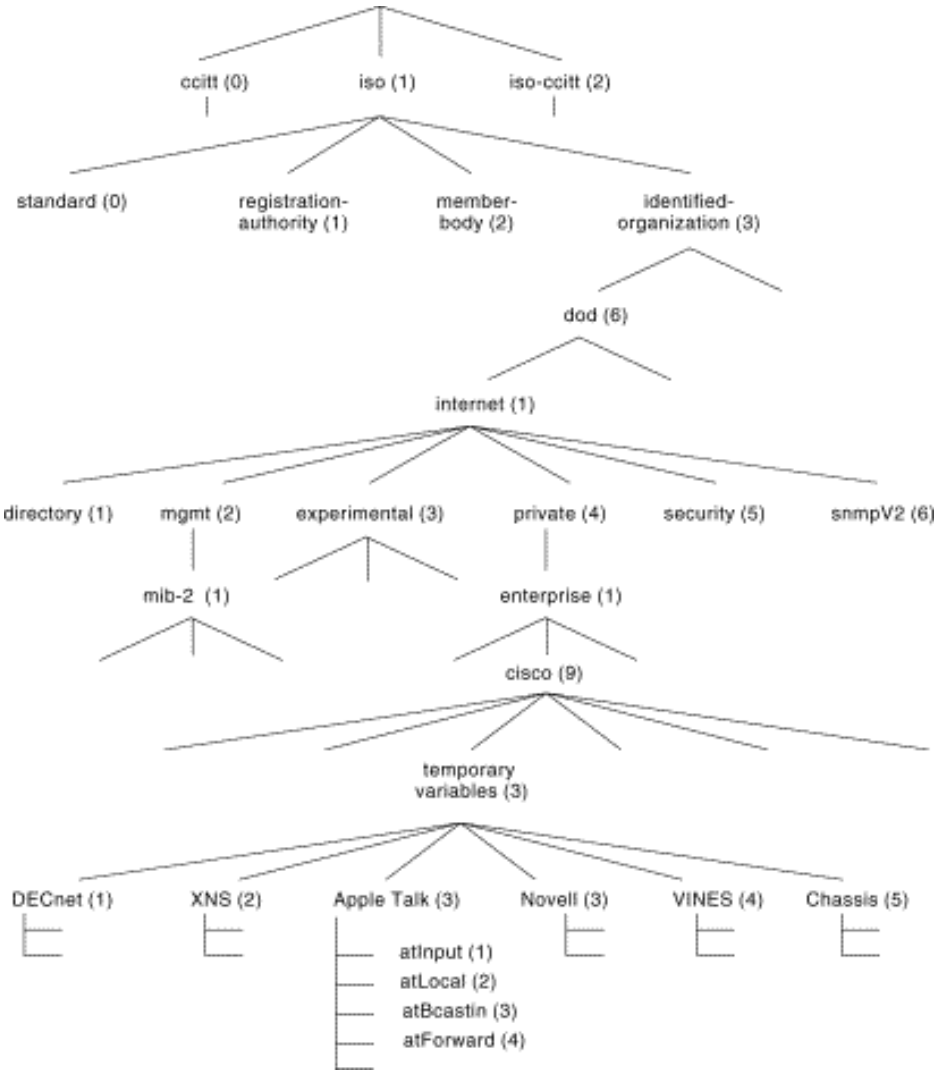
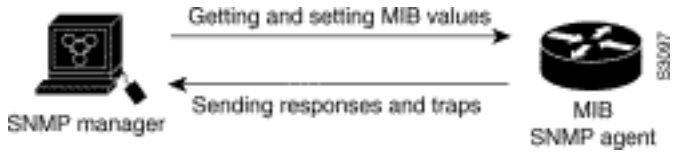


Fig. 6. La struttura della MIB

Sia l'agente che il manager per poter comunicare possiedono una base di conoscenza comune che prende il nome di MIB. Una Management Information Base è una collezione di informazioni organizzate gerarchicamente chiamate OID [OID] (Object Identifiers). Gli OID possono essere di due tipi: scalari, che definiscono una singola

istanza di oggetto, oppure tabulari che rappresentano una moltitudine di istante di oggetto raggruppate in tabelle.

Un OID è identificato unicamente dal suo nome in forma gerarchica oppure dal suo descrittore equivalente. Come si vede nella figura a fianco l'OID .iso.identified-organization.dod.internet.private.enterprise.cisco.temporaryvariables.AppleTalk.atInput corrisponde all'Object ID numerico .1.3.6.1.4.1.9.3.3.1.
Gli OID foglia contengono il rispettivo valore dell'oggetto, allo stesso modo per cui in un linguaggio di programmazione ad una variabile è assegnato un valore.
La comunicazione tra manager e agente avviene per mezzo di semplici comandi SNMP che permettono la lettura (comandi Get, GetNext, GetBulk) e la scrittura (comando Set) della coppia OID/valore. Ci sono poi altre forme di trasmissione asincrona chiamate notifiche SNMP. Questo tipo di informazioni possono essere trasmesse come trap o inform.



Facciamo qualche esempio con una macchina linux e un router Cisco.
Installiamo il manager sulla linux box e abilitiamo l'agente sul router (versione 2c, community stage) con il comando:

```
router(config)#snmp-server community stage
```

Ora dalla console del manager richiediamo il nome del dispositivo monitorato (comando Get):

```
$ snmpget -v 2c -c stage cisco system.sysDescr.0
SNMPv2-MIB::sysDescr.0 = STRING: Cisco Internetwork Operating System Software IOS (tm) C800 Software (C800-Y6-MW), Version 12.0(5)T,  RELEASE SOFTWARE (fc1) Copyright (c) 1986-1999 by cisco Systems, Inc. Compiled Fri 23-Jul-99 01:30 by kpma
```

Assegniamo all'OID sysContact il nome dell'amministratore del router (admin) con il comando Set:

```
$ snmpset -v 2c -c stage cisco system.sysContact.0 s admin
SNMPv2-MIB::sysContact.0 = STRING: admin
$ snmpget -v 2c -c stage cisco system.sysContact.0
SNMPv2-MIB::sysContact.0 = STRING: admin
$ snmptranslate -On -IR sysContact.0.1.3.6.1.2.1.1.4.0
```

Osserviamo che l'OID sysContact.0, presente nella MIB SNMPv2-MIB con identificativo .1.3.6.1.2.1.1.4.0, è di tipo stringa e vale "admin".

Il concetto di notifiche SNMP è leggermente diverso: al contrario dei comandi Get/Set, in cui la trasmissione SNMP tra agent e manager è sincrona, il router invia notifiche quando si verificano particolari eventi: condizioni d'errore, superamento di una soglia RMON, log via syslog, etc…

Per la scrittura di RIDS ho utilizzato una implementazione Opensource (con licenza BSD [BSD]) di un NMS per sistemi POSIX (Unix-Like) e Windows: Net-SNMP [NSNMP]. Questo pacchetto fornisce i comandi SNMP visti sopra e una libreria per gestire il protocollo SNMP da un programma scritto in C.

7.2 **L'architettura di RIDS**

In questo paragrafo andrò ad illustrare brevemente l'architettura software di RIDS.

Come detto poco prima alla fine del capitolo 6, RIDS non è stato pensato per essere utilizzato in una struttura informativa di produzione, ma come **esempio di Router-IDS** che utilizza e valorizza l'approccio context-based. L'implementazione di RIDS non offre molte delle feature proposte in fase di progettazione, tuttavia nei paragrafi a seguire non faccio distinzione tra caratteristiche presenti/mancanti nel codice, la possibilità di continuarne lo sviluppo è offerta a chiunque :)

L'architettura di RIDS è decisamente semplice e rispecchia il modello generico di IDS discusso nel secondo capitolo. Sono definiti quattro **macro-componenti**:

- Input: recupera la configurazione corrente del router attraverso SNMP[1] (è previsto il supporto futuro di RMON [RM] e Cisco Net-Flow [CNF][2]). Il termine "configurazione" ha una valenza più ampia di quello comunemente utilizzato per indicare il contenuto dei file di personalizzazione di un'applicazione: fa riferimento allo stato attuale di funzionamento dell'apparato, che comprende, oltre alla configurazione del sistema operativo[3], lo stato delle variabili hardware (temperatura, carico della CPU, etc.) e software (cache arp, tabella di routing, etc.) significative per il dispositivo. Applicando i principi della "conoscenza a priori" e del "focus", la "configurazione" utilizzata da RIDS è unica e strettamente dipendente dalle caratteristiche del router (marca e modello del router, tipo e versione del sistema operativo, componenti hardware presenti, funzionalità software disponibili e abilitate, contesto di utilizzo del router, etc..).

  Il componente di Input utilizza la modalità attiva e passiva: in quella attiva il manager SNMP richiede le informazioni all'agent del router[4] con una query SNMP; in quella passiva è il router a trasmettere la configurazione a RIDS sotto forma di trap SNMP.

[1] Per una questione di portabilità e flessibilità.
[2] Altri due protocolli di controllo remoto.
[3] Archiviata nel caso di Cisco IOS nel file "running-config".
[4] L'agent SNMP deve essere abilitato sul router. Il comando IOS per SNMPv2 è `#snmp-server community nome_community.`

- Archiviazione: implementa i meccanismi di accesso in lettura e scrittura ai dati archiviati nel database di history (le configurazioni del router e gli eventi generati dall'IDS).
- Core: è il componente più complesso e delicato che realizza tutte le le funzioni di Intrusion Detection. Ogni famiglia di IDS implementa meccanismi di identificazione delle anomalie e delle intrusioni particolari per quel target-system (un router-IDS ha un componente di Core veramente diverso da un firewall-IDS in quanto un router e uno firewall svolgono due funzioni completamente diverse!). Inoltre le procedure di analisi e di detection sono personalizzate sulla base della "configurazione"[1] del target-system; in ogni modo per tutti gli IDS queste funzioni si classificano in tre categorie:
    - anomaly based: mettono a confronto tra la configurazione corrente e con quella corretta, appresa nella fase di tuning (si veda sotto)
    - storiche: confrontano la configurazione corrente con quelle passate
    - di correlazione: correlano parametri diversi della stessa configurazione temporale
- Comunicazione: gestisce la trasmissione degli eventi e soddisfa le richieste di informazioni dell'IDS di livello superiore. Con il termine "IDS di livello superiore" si intenta l'IDS delegato al controllo della community o dello scenario di appartenenza del Router-IDS.

RIDS implementa due **modalità** di funzionamento:

1) Tuning: in questa modalità l'IDS si preoccupa di:
    - Analizzare e adattarsi al contesto di operatività dell'IDS tramite le seguenti operazioni:
        - identificare il tipo di router: RIDS recupera l'hostname, la versione del sistema operativo, il numero, il tipo e lo stato funzionamento delle interfacce e altro;
        - verificare l'esistenza di feature opzionali: i servizi base di un router possono essere estesi tramite aggiornamenti del sistema operativo e l'aggiunta di

[1] Si veda la definizione del termine "configurazione" data all'interno del componente di Input.

moduli opzionali che prendono il nome di “Features Set” (per esempio IOS Firewalling + IOS IDS). In questa fase RIDS costruisce un profilo della configurazione software del router. In particolare questo processo è implementato verificando il valore di ritorno a query SNMP di Object ID non-standard;

- personalizzare l'attività di Intrusion Detection: RIDS permette all'amministratore di personalizzare il controllo sul router andando a scegliere in dettaglio cosa monitorare. RIDS propone un profilo di scelte di default in funzione delle informazioni recuperate dalle due fasi di tuning precedenti. Per esempio escluderà di default dal controllo le interfacce il cui stato amministrativo è “down”[1].

Al termine di questa fase RIDS definisce la “configurazione”[2] ideale per il router.

- Costruire un modello di funzionamento corretto del router: fase in cui RIDS monitora passivamente le attività del router ed elabora un profilo di funzionamento corretto del target-system. Questa attività è estremamente delicata e importante per tutti gli IDS anomaly based. In letteratura sono stati proposti numerosi algoritmi più o meno precisi per modellare il comportamento di un sistema non predicibile[3]. RIDS utilizza un criterio molto semplice poiché, come ho già detto, il fine di questo strumento è solamente quello di mostrare concretamente le caratteristiche e le potenzialità dell'approccio context-based e non quello di produrre un software utilizzabile in un contesto produttivo.

RIDS funziona in modalità di tuning alla prima esecuzione e successivamente a richiesta dell'amministratore. E’ consigliabile rieseguire la fase di tuning per riallineare la configurazione di RIDS a quella dell’apparato e della rete, qualora siano state fatte delle modifiche.

2) Normale: nella modalità di funzionamento normale RIDS utilizza un approccio simile a quello di un generico Intrusion Detection System. Periodicamente il

[1] A tal proposito si veda il paragrafo 7.3.1
[2] Il termine “configurazione” assume un signifacato più ampio, vedi sopra.
[3] Si è già parlato di questo aspetto all'interno del capito dedicato alla tassonomia degli IDS.

componente di Core elabora il flusso di informazioni dell'Input generando possibili allarmi per il SSO e mantenendo aggiornata la base dati di history. In più, in funzione della possibilità di appartenere ad una community o ad uno scenario, invia all'IDS appropriato gli eventi utili al contesto di community/scenario. Viceversa l'IDS di livello superiore può richiedere a RIDS delle informazioni precise contenute nella base dati di History.

Nella progettazione di un IDS context-based, una fase molto importante è la definizione della configurazione da utilizzare nell'analisi del sistema controllato. Nel caso specifico di RIDS, questa attività consiste nell'identificazione delle "variabili" significative di un router, che caratterizzano il funzionamento di quel particolare dispositivo di rete. La fase implementativa traduce la configurazione appena definita in strutture dati conforme al protocollo SNMP. Più precisamente ogni "variabile" corrisponde a un Object ID (OID)[1], il cui elenco è riportato nel paragrafo successivo.

[1] SNMP è stato descritto nel paragrafo 7.1

7.3 **L'implementazione**

Nella fase implementativa la configurazione del router è tradotta in un elenco di **Object ID**, ciascuno definito da un file di Management Information Base.
Le MIB sono classificabili in due categorie:

- standard: le MIB di questa categoria sono comuni a tutti gli appartati di rete e possono essere utilizzate router di marche e modelli differenti. Fan parte di questa categoria le MIB inerenti la configurazione SNMP, delle interfacce e dei protocolli ip, tcp, udp e icmp.
- specifici: queste MIB controllano funzionalità e servizi particolari che non tutti i router possiedono. Cisco fornisce un elenco completo delle MIB supportate in funzione del modello e del sistema operativo del router.

  Tutte le MIB che contengono la stringa OLD sono compatibili con SNMP v1, tutte le altre funzionano con SNMP v1 e v2. Le MIB che iniziano per CISCO fan parte di quell'albero di MIB riservato al omonimo vendor; è presumibile che siano supportate solo dai router Cisco.

  Per esempio, la versione 12.3(1) di Cisco IOS (con una Features Set IP/FW/IDS)[1] supporta MIB quali CISCO-CAR-MIB, CISCO-CONFIG-COPY-MIB, CISCO-CONFIG-MAN-MIB, CISCO-ENVMON-MIB, CISCO-MEMORY-POOL-MIB, IF-MIB, OLD-CISCO-CPU-MIB, RFC1231-MIB, SNMP-TARGET-MIB, SNMP-USM-MIB, SNMP-VACM-MIB, SNMPv2-MIB, TCP-MIB, UDP-MIB, etc...

I paragrafi a seguire sono un **elenco strutturato delle "variabili" (e corrispondenti OID)** utilizzati nell'implementazione di RIDS. Ho lasciato alcune descrizioni in inglese: sono prese direttamente dal file di definizione[2] della MIB.

7.3.1 Problemi sulle interfacce

Lo stato operativo

Ogni interfaccia possiede due variabili che definiscono lo stato operativo dell'interfaccia: uno definito dall'amministratore e uno corrente. Pare ovvio che se lo

[1] Il router utilizzato in testing è un Cisco 2621, versione di IOS 12.3(1) e Feature Set IP/FW/IDS. L'immagine del sistema operativo è c2600-io3-mz.12.3-1.
[2] Cisco fornisce i file di definizione delle MIB supportate dai suoi device. Sono reperibili dal sito.

stato amministrativo e operativo non coincidono ci siano dei problemi sull'interfaccia. Questo può succedere o perché l'interfaccia perde il link o la scheda si rompe. In questi casi lo stato amministrativo rimane up(1) e quello operativo va giù(2). RIDS effettua controlli sulla discordanza dei due stati con gli OID di cui sotto.

La MIB IF-MIB definisce a tal proposito due Object ID:

- ifAdminStatus: "The desired state of the interface. The testing(3) state indicates that no operational packets can be passed. When a managed system initializes, all interfaces start with ifAdminStatus in the down(2) state. As a result of either explicit management action or per configuration information retained by the managed system, ifAdminStatus is then changed to either the up(1) or testing(3) states (or remains in the down(2) state)."
- ifOperStatus: “The current operational state of the interface. The testing(3) state indicates that no operational packets can be passed. If ifAdminStatus is down(2) then ifOperStatus should be down(2). If ifAdminStatus is changed to up(1) then ifOperStatus should change to up(1) if the interface is ready to transmit and receive network traffic; it should change to dormant(5) if the interface is waiting for external actions (such as a serial line waiting for an incoming connection); it should remain in the down(2) state if and only if there is a fault that prevents it from going to the up(1) state; it should remain in the notPresent(6) state if the interface has missing (typically, hardware) components."

La modalità promiscua

Alcune interfacce di rete, come quelle ethernet per esempio, hanno la possibilità di funzionare in modalità promiscua in modo da accettare anche i frame non destinati a sè (quelli in cui l'indirizzo hardware di destinazione è diverso dal proprio e da quello di broastcast/multicast). In tal caso un attaccante, utilizzando uno sniffer è in grado di catturare da zona utente tutto il traffico passante per una interfaccia promiscua. Questo pericolo è ancora più sentito nelle reti hubbate dove a ogni nodo arrivano fisicamente tutti i pacchetti circolanti, sia i propri che quelli dei vicini.

- La MIB relativa alle interfacce definisce l'OID ifPromiscuousMode che vale 1 quando l'interfaccia è in modalità promiscua.

Link trap

La IF-MIB definisce una trap per il controllo del link dell'interfaccia. L'OID di riferimento è ifLinkUpDownTrapEnable: "Indicates whether linkUp/linkDown traps should be generated for this interface. By default, this object should have the value enabled(1) for interfaces which do not operate on 'top' of any other interface (as defined in the ifStackTable), and disabled(2) otherwise."

Pacchetti scartati dall'interfaccia o dallo stack tcp-ip

Alcuni Object ID sono utilizzati da RIDS per osservare il traffico anomalo verso il router, che viene scartato dal driver dell'interfaccia o dallo stack di rete. In particolare le MIB inerenti le interfacce e i protocolli TCP/IP definiscono:

- ifInDiscard: "The number of inbound packets which were chosen to be discarded even though no errors had been detected to prevent their being deliverable to a higher-layer protocol. One possible reason for discarding such a packet could be to free up buffer space. Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ifCounterDiscontinuityTime."
- ifInErrors: “For packet-oriented interfaces, the number of inbound packets that contained errors preventing them from being deliverable to a higher-layer protocol. For character- oriented or fixed-length interfaces, the number of inbound transmission units that contained errors preventing them from being deliverable to a higher-layer protocol.”
- ifInUnknownProtos: “For packet-oriented interfaces, the number of packets received via the interface which were discarded because of an unknown or unsupported protocol. For character-oriented or fixed-length interfaces that support protocol multiplexing the number of transmission units received via the interface which were discarded because of an unknown or unsupported protocol. For any interface that does not support protocol multiplexing, this counter will always be 0.”
- ipInDiscards: “The number of input IP datagrams for which no problems were encountered to prevent their continued processing, but which were discarded (e.g., for lack of buffer space). Note that this counter does not include any datagrams discarded while awaiting re-assembly."
- ipInHdrErrors: “The number of input datagrams discarded due to errors in their

IP headers, including bad checksums, version number mismatch, other format errors, time-to-live exceeded, errors discovered in processing their IP options, etc."

- ipInUnknownProtos: "The number of locally-addressed datagrams received successfully but discarded because of an unknown or unsupported protocol."
- tcpInErrs: "The total number of segments received in error (e.g., bad TCP checksums)."
- udpInErrors: "The number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port."
- icmpInErrors: "The number of ICMP messages which the entity received but determined as having ICMP-specific errors (bad ICMP checksums, bad length, etc.)."

7.3.2 Attacchi DoS

Molto spesso i router, specialmente quelli perimetrali, sono vittima di attacchi di Deny of Service, mirati principalmente a negare un servizio di connettività. Esistono molti appliance NIDS che utilizzando l'approccio anomaly-based costruiscono un profilo di traffico normale e rilevano con precisione attacchi DoS e Worms. (Gli worm generano un gran volume di traffico, basti pensare a worm che si diffondono per mail!). Questi dispositivi hanno un costo di vendita molto elevato che solamente grosse aziende possono sostenere: questo tipo di soluzioni sono adottare a livello di Core per dorsali e grandi reti d'accesso.
RIDS implementa alcuni semplici controlli relativi ai DoS syn-flood e icmp-flood.

syn-flood

Questo attacco è realizzato sfruttando una vulnerabilità intrinseca del protocollo TCP: l'attaccante invia numerosi pacchetti TCP con il flag SYN settato a 1[1] a una porta aperta della macchina vittima. In questo scenario, l'host ricevente risponde con un pacchetto con i flag SYN e ACK settatati e alloca le risorse per gestire la connessione (il socket). A questo punto l'attaccante non conclude la normale procedura di 3-handshake con un

[1] In accordo con la procedura di 3-handshaking, un pacchetto TCP/IP con il solo flag SYN settato sta ad indicare una richiesta di inizio connessione.

pacchetto ACK, ma evita di rispondere. Si può fingersi un host inesistente o mettere un filtro sulla macchina attaccante tale da impedire al kernel del sistema operativo di rispondere al SYN|ACK in arrivo dalla macchina vittima con un RST (il kernel non avrà visto il nostro primo pacchetto SYN e quindi segnerà il pacchetto in arrivo come invalido). In questo modo la macchina vittima istanza un numero sempre maggiore di connessioni attive che non utilizzerà, in cui lo stato del socket rimane fino allo scadere di un timeout nello stato di SYN-RCVD. Questa situazione esaurisce le risorse di memoria del router obbligandolo al crash. Maggiori informazioni in rete, per esempio qua [SYNF].

- A grandi linee si può utilizzare l'OID tcpAttemptFails come variabile caratterizzante: "The number of times TCP connections have made a direct transition to the CLOSED state from either the SYN-SENT state or the SYN-RCVD state, plus the number of times TCP connections have made a direct transition to the LISTEN state from the SYN-RCVD state."

Questo indicatore non è molto preciso in quanto tiene in considerazione anche il passaggio SYN-SENT->CLOSED. Tuttavia se non ci sono collegamenti dalla shell del router all'esterno tale valore è pari a 0 e l'algoritmo di detection funziona correttamente.
RIDS utilizza lo stesso OID per identificare attività di port-scanning di tipo half-open (vedi paragrafo 7.3.4).

Dopo aver condotto una serie di test sperimentali con strumenti comunemente utilizzati per realizzare attacchi di questo tipo (hping[HP] e neptune.c[1]) ho valutato una soglia di allarme a 100 pacchetti/sec.

icmp-flood

L'attacco icmp-flood è il DoS più facile da realizzare e consiste in una saturazione della banda per mezzo di icmp echo-request (ping). La MIB standard relativa al protocollo IP (IP-MIB) definisce alcuni OID per il controllo dei pacchetti ICMP:

- icmpInEchos: numero di icmp echo-request ricevuti dal router.

[1] http://www.phrack.org/phrack/48/P48-13.

Alcune prove con hping e ping[1] hanno suggerito una soglia di allarme di circa 100 ping/sec.

7.3.3 Volume di traffico anomalo

Attraverso SNMP è possibile costruire un profilo normale del traffico passante per il router in modo da identificare, attraverso una procedura anomaly-based, variazioni significative dovute alla presenza di worm o di spyware. Ho già accennato della difficoltà intrinseca di tracciare un comportamento normale di una variabile fisica non predicibile a priori: in letteratura sono stati proposti molti modelli matematici, alcuni dei quali sono riportati in bibliografia.
RIDS, giusto a titolo d'esempio, costruisce un semplice profilo basato su medie orarie. La fase di campionamento deve essere eseguita prima di quella operativa dell'IDS, e richiede un numero di giorni sufficientemente ampio (intorno a un mese). Il profilo è tracciato per interfaccia (ma non per protocollo) facendo un rapporto incrementale tra le variabili ifInOctets e ifOutOctets e il tempo:

- ifInOctets: “The total number of octets received on the interface, including framing characters.”
- ifOutOctets: “The total number of octets transmitted out of the interface, including framing characters.”

7.3.4 Information gathering via portscanning

Un portscan è una procedura di information gathering con cui è possibile identificare da remoto i servizi (tipo e versione) offerti da un host e contemporaneamente valutare la presenza di un firewall. Esistono molte tecniche di portscanning, più o meno raffinate. Nel mondo unix-like lo strumento di riferimento è nmap [NMAP] nella cui manpage[2] sono elencate e descritte le modalità di portscanning supportate (non è il fine di questo documento quello di descrivere le tecniche di portscanning).

Come tutti gli strumenti di sicurezza, il risultato ottenuto deve sempre essere preso con le pinze: RIDS suggerisce la possibilità di un'azione di portscanning, ma è l'amministratore che deve valutare con precisione la situazione facendo un'indagine più

[1] Passando l’opzione –f a ping otteniamo un effetto di flooding.
[2] Termine tecnico per indicare le pagine di documentazione di un programma.

approfondita.

Ci sono due variabili nella MIB relativa al TCP utilizzabili come indice:

- tcpAttemptFails: "The number of times TCP connections have made a direct transition to the CLOSED state from either the SYN-SENT state or the SYN-RCVD state, plus the number of times TCP connections have made a direct transition to the LISTEN state from the SYN-RCVD state."
- tcpOutRsts: "The number of TCP segments sent containing the RST flag.

Semplificando, tcpAttemptFails è un contatore di 3-handshake non completati. Questa situazione si verifica, come descritto precedentemente nel paragrafo relativo ai syn-flood, qualora al router non arrivi l'ACK di risposta al SYN|ACK mandato precedentemente. Nel caso di un portscanner di tipo half-open, eseguibile passando il flag -sS a nmap, l'attaccante invia appositamente un RST di riposta al SYN|ACK del router in modo da bloccare la procedura di inizializzazione della connessione e conseguentemente andando a incrementare la variabile tcpAttemptFails monitorata dall'IDS.

L'Object ID tcpOutRsts tiene traccia dei pacchetti TCP con flag RST impostato generati in uscita dal router. Il protocollo TCP/IP ammette l'utilizzo di tale flag per resettare una connessione o per rispondere ad un pacchetto non valido ricevuto. Nella pratica, tutto dipende dall'implementazione dello stack tcp/ip del sistema operativo, infatti in alcune situazioni è preferibile ignorare i pacchetti errati piuttosto che rispondere con un RST. Altre volte questa flag è utilizzata spropositatamente in altre situazioni. Nello specifico, Cisco IOS risponde con un RST qualora riceva una richiesta di connessione o pacchetti con combinazioni inusuali di flag su porte chiuse. Conducendo qualche prova con nmap ho notato che tale situazione si verifica qualora si effettui un portscan syn-half, stealth fin, xmas tree e null (vedasi la manpage per la descrizione del tipo di portscan).

Analogamente, RIDS identifica correttamente i portscan UDP controllando il numero di datagrammi UDP ricevuti al secondo verso porte chiuse:

- udpNoPorts: "The total number of received UDP datagrams for which there was no application at the destination port."

7.3.5 Attacchi alla routing table

Il processo di routing mantiene aggiornata e coerente con le informazioni provenienti dagli altri router una routing table (RT), che viene utilizzata dal processo di forwarding nell'operazione di routing dei pacchetti ricevuti dalle interfacce del router.
La routing table è una tabella composta da una serie di righe, in cui ciascuna riga definisce la destinazione futura (il next-hop), la metrica e l'interfaccia associata per una specifica rotta. Ciascun nodo di una rete possiede una RT che viene utilizzata dal stack di rete del kernel come mappa per i pacchetti in uscita.
Facciamo un esempio con un semplice client:

```
Kernel IP routing table
Destination   Gateway         Genmask       Flags Metric Ref Use Iface
192.168.1.0   0.0.0.0         255.255.255.0   U     0      0   0  eth0
127.0.0.0     0.0.0.0         255.0.0.0       U     0      0   0  lo
0.0.0.0       192.168.1.100   0.0.0.0         UG    0      0   0  eth0
```

Ogni riga è composta da vari campi tra cui:

- Destination: la destinazione o rotta;
- Gateway: il next-hop verso cui mandare i pacchetti;
- Genmask: la netmask associata alla rotta;
- Iface: l'interfaccia in cui è raggiungibile il next-hop;
- Flags:
  - U: la rotta è su;
  - H: la rotta è un host;
  - G: la rotta una un gateway

Ora è facile osservare che le prime due righe definiscono la classe di indirizzi associati alle interfacce di loopback e di rete (eth0), mentre l'ultima specifica che ogni destinazione è raggiungibile attraverso il next-hop con indirizzo 192.168.1.100 raggiungibile sull'interfaccia eth0. Il confronto dell’indirizzo di rete del pacchetto con le rotte della routing table procede con la ricerca di una rotta di tipo host, poi di tipo subnet e infine di tipo gateway (rotta pari a 0.0.0.0). Per cui un pacchetto viene mandato al gateway solo se la sua destinazione non è all'interno delle due sottoreti definite dalle prime due rotte.

La RT può contenere due tipi di rotte: quelle statiche si mantengono per tempo illimitato, e quelle dinamiche si auto-adattano alla configurazione e alla dinamicità della rete. Il processo di routing gestisce le rotte dinamiche attraverso l'uso dei protocolli di routing dinamici che permettono a più router, presenti nella stessa rete per esempio, di scambiarsi le proprie RT cosicché tutti i router abbiano una RT coerente con la tipologia fisica della rete.
L'uso dei protocolli di routing dinamico offre molti vantaggi tra cui:

- l'amministratore è assolto dal problema di scrivere una RT statica per ogni router
- se un router è congestionato o cade (si rompe), o ne viene inserito uno nuovo, le routing table di tutti i router si adattano automaticamente all'evento.

Solitamente i protocolli di routing dinamico sono supportati dai router e non dai dispositivi general purpose come i normali personal computer. L'esigenza di collegare numerosi router per permettere lo scambio di dati tra sottoreti diverse, ha spinto alla necessità di creare dei protocolli che fossero in grado di “leggere” la tipologia della rete e di adattare la configurazione della routing table ai suoi cambiamenti.

La routing table riportata qui sotto mostra l'utilizzo del protocollo di routing dinamico RIP per un router con sistema operativo Cisco IOS:

```
Router#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate default
       U - per-user static route

Gateway of last resort is not set
C       192.168.2.0/24 is directly connected, 192.168.2.0
C       192.168.1.0/24 is directly connected, 192.168.1.0
R       192.168.4.0/24 [120/1] via 192.168.1.1, 00:06:26, Ethernet0
R       192.168.3.0/24 [120/1] via 192.168.2.1, 00:04:37, Ethernet1

Router#ping 192.168.3.0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.3.0, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
```

Le prime due righe della RT mostrano le subnet assegnate alle due interfacce di rete nello stesso modo della RT del client Linux (lo si vede dal flag C=connect).

Le altre due, sono state create dinamicamente dal protocollo RIP (flag R) e hanno un tempo di timeout totale pari rispettivamente a 6 e 4 minuti. La prima rotta è raggiungibile attraverso il next-hop con indirizzo ip 192.168.1.1 sull'interfaccia Ethenet0 e con metrica 1, la seconda attraverso 192.168.2.1 sull'interfaccia Ethernet1 con metrica 1.

Esistono due grosse famiglie di routing protocols: quelli che utilizzano gli algoritmo di link-state e quelli che si appoggiano agli algoritmi distance-vector o Bellman-Ford.
Ciò che differenzia i due algoritmi è che i protocolli appartenenti alla prima famiglia mandano update di piccole porzioni della RT, solamente i link a cui sono attaccati, a tutti i nodi della rete; mentre i secondi inoltrano l'intera RT ma solo ai router vicini.
Inoltre ogni routing protocol utilizza una *metrica* diversa per indicare la qualità di una rotta. Il processo di routing utilizza la metrica di ogni pacchetto di update per stimare il percorso migliore per una destinazione così da tenere aggiornata la RT con i next-hop migliori per ogni destinazione.
RIP, protocollo distance-vector, utilizza una metrica fisica: il numero di hop (nodi) che un pacchetto spedito dalla sorgente deve attraversare prima di arrivare a destinazione. Nell'esempio di cui sopra le due rotte dinamiche sono raggiungibili entrambe con metrica uno, ciò vuol dire che la destinazione è direttamente raggiungibile da una delle interfacce del next-hop.
Altri tipi di protocolli come EIGRP utilizzano il termine *costo* per attribuire ad una rotta un valore di qualità in funzione di più metriche. Più in generale alcune metriche sono la lunghezza di un canale (path length), la larghezza di banda (bandwidth), il ritardo di comunicazione (routing delay), il carico sui router (load) e la sicurezza del canale (reliability).

Un Router-IDS deve essere in grado di monitorare i cambiamenti della routing table del router apportati dal processo di routing. Il processo di routing, che si preoccupa di aggiornare la RT in funzione degli update provenienti dalla rete, utilizza algoritmi diversi in funzione del protocollo di routing utilizzato. RIPv2, che utilizza la metrica più banale dell'hop-count, si appoggia al classico algoritmo distance-vector qui brevemente descritto:

- ciascun router spedisce la propria routing-table in multicast per mezzo di pacchetti di update ogni 30 secondi o quando la propria RT cambia

- alla ricezione di un messaggio di update il processo di routing verifica la correttezza semantica dell'informazione (la rotta non sia del tipo 127, il next-hop non sia un indirizzo di broadcast, la metrica non sia maggiore di infinito (15 per rip)).
- se il test fallisce il pacchetto viene scartato altrimenti la metrica viene incrementata di 1 e la rotta è confrontata con la RT:
  - se la rotta non esiste viene inserita nella RT
  - se la rotta è presente con una metrica peggiore (più grande) o con metrica uguale ma netmask maggiore viene accettata
  - se la rotta è presente e il next-hop è il mittente del messaggio di update la rotta viene modificata qualora la metrica sia differente.

Ma che necessità c'è di verificare gli aggiornamenti della routing-table? Molti protocolli di routing non implementano meccanismi di autenticazione forti[1] e quindi un attaccante è in grado di creare update ad hoc tali da modificare a proprio piacimento la routing-table del router vittima. In questo modo l'attaccante può facilmente sostituire il next-hop di una rotta con sé stesso per inserirsi nel flusso dati desiderato e recuperare informazioni riservate. Oppure è in grado di modificare il default gateway dei router interni verso una destinazione irraggiungibile così da impedire l'accesso ad Internet (attacco DoS). Un tool d'esempio è RIPPER [RIPPER] che avvelena la tabella di routing attraverso pacchetti RIP costruiti ad-hoc. Esistono molti scenari d'attacco, l'unico limite è la fantasia dell'attaccante :-) L'autore e altri due membri del gruppo italiano S.P.I.N.E. [SPINE] hanno tenuto un dibattito sull'argomento la cui presentazione è disponibile sul Web [ASPRD].

Nell'implementazione di RIDS è stato previsto un meccanismo futuro di monitoring e di verifica della correttezza degli aggiornamenti della RT, in prima istanza RIP essendo il protocollo più semplice. L’algoritmo è ancora in fase embrionale per cui preferisco non riportarlo in questo documento ma affrontarlo in un’altra sede.
La tabella ipCidrRouteTable della MIB IP-FORWARD-MIB contiene tutta la routing table del router:

- ipCidrRouteTable: “The IP CIDR Route Table obsoletes and replaces the

[1] Per esempio, RIP nelle versioni 1 e 2, inseriscono la chiave di autenticazione in formato plain-text all’interno del proprio header.

ipRoute Table current in MIB-I and MIB-II and the IP Forwarding Table. It adds knowledge of the autonomous system of the next hop, multiple next hops, and policy routing, and Classless Inter-Domain Routing.

7.3.6 Attacchi a CDP

Cisco Discovery Protocol è un protocollo a layer 2 implementato su qualsiasi dispositivo Cisco tra cui router, access server, bridge e switch. Attraverso CDP, un dispositivo informa i "vicini" della sua esistenza ogni 30 secondi (di default) e allo stesso tempo riceve informazioni dagli altri, cosicché ogni dispositivo conosca le periferiche vicine.
I messaggi CDP contengono informazioni del dispositivo come l'hostname, le interfacce di collegamento, la versione e il tipo di sistema operativo e l'indirizzo IP.
Il protocollo CDP non è autenticato e quindi è possibile, da parte di un attaccante, spoffare pacchetti CDP per alterare le informazioni possedute dai router suoi vicini o per consumare risorse di memoria sul router vittima attraverso un semplice DoS che cerca di aggiungere entry inesistenti.
Il gruppo tedesco Phenoelit ha sviluppato un tool di esempio all'interno della suite Irpas [IRPAS] che dimostra le vulnerabilità del protocollo.

RIDS verifica lo stato CDP sulle interfacce e genera un allarme quando CDP è abilitato. In tal caso RIDS monitora i cambiamenti della cache CDP. Al momento dell'inserimento o della modifica di un vicino, RIDS verifica la correttezza dell'aggiornamento confrontando le informazioni CDP con quelle reali, prelevate dall'IDS che monitora il device da cui proviene il messaggio CDP (se esiste)[1] oppure ricercate per mezzo di scansioni opportune della rete.
I vantaggi nell'uso di CDP sono talmente irrisori che la disattivazione del protocollo è una soluzione d'obbligo.

Controllo configurazione protocollo CDP: CISCO-CDP-MIB.my

- Verifica abilitazione CDP sulle interfaccie: cdpInterfaceTable: "The (conceptual) table containing the status of CDP on the device's interfaces."

[1] Nell'architettura distribuita del Router-IDS è previsto il trasferimento della gestione di un controllo dall'IDS di livello più basso a quello di livello scenario/comunità. Nel caso di CDP sarà l'IDS dello scenario che verificherà la correttezza delle cache CDP tra i vari dispositivi.

- Verifica aggiornamenti: cdpCacheTable: "The (conceptual) table containing the cached information obtained via receiving CDP messages."

7.3.7 Attacchi SNMP[1]

Attraverso il protocollo SNMP un attaccante può recuperare numerose informazioni del router vittima: l'hostname, lo stato dei socket, la tabella di routing, i protocolli abilitati e molto altro. Inoltre, se la community ha privilegi di scrittura, può modificare remotamente la configurazione del router.
Ovviamente l'attaccante deve conoscere la community impostata nell'agent SNMP, che di default è public ma può essere cambiata. SNMP v1 e v2 trasmettono la community in plain text in rete, per cui è sufficiente un'azione di sniffing per recuperarla. SNMP v3 invece implementa una serie di algoritmi di autenticazione e di hashing che rendono più sicuro l'utilizzo del protocollo. Tuttavia un attaccante può sempre recuperare le credenziali di login attraverso un attacco brute force verso il router.

RIDS utilizza una serie di variabili definite in SNMPv2-MIB per controllare gli eventi SNMP, in particolare:

- snmpInBadVersions: "The total number of SNMP messages which were delivered to the SNMP entity and were for an unsupported SNMP version."
- snmpInBadCommunityNames: "The total number of SNMP messages delivered to the SNMP entity which used a SNMP community name not known to said entity."
- snmpInBadCommunityUses: "The total number of SNMP messages delivered to the SNMP entity which represented an SNMP operation which was not allowed by the SNMP community named in the message."
- snmpInASNParseErrs: “"The total number of ASN.1 or BER errors encountered by the SNMP entity when decoding received SNMP messages."
- snmpEnableAuthenTraps: "Indicates whether the SNMP entity is permitted to generate authenticationFailure traps. The value of this object overrides any configuration information; as such, it provides a means whereby all authenticationFailure traps may be disabled.”

[1] Per una descrizione del protocollo di veda il paragrafo 7.2

7.3.8 Alto numero di connessioni

Praticamente tutti i router permettono di effettuare un login via telnet da rete anche se è caldamente consigliabile utilizzabile la porta seriale, evitando di lasciare il servizio di telnet abilitato. Un attaccante potrebbe portare un attacco di brute-force mirato al recupero delle credenziali di accesso alla console, oppure sferrare un DoS effettuando molteplici collegamenti simultanei al servizio di telnet. Ovviamente questo discorso è estendibile ad altri servizi di scarsa importanza o evitabili.
Viceversa è possibile accedere ad un servizio esterno dalla shell del router. Ottenere delle credenziali di accesso corrette significa aver la possibilità di accedere a una seconda macchina mascherando il proprio indirizzo IP con quello del router vittima. Un controllo periodico dello stato dei socket delle connessioni e della configurazione del protocollo TCP permette a RIDS di identificare queste condizioni di anomalia.

Nella MIB TCP-MIB:

- numero di connessioni dal router all'esterno: tcpActiveOpens: "The number of times TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state.
- numero di connessioni con il router: tcpCurrEstab: "The number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT.”

7.3.9 Modifica delle configurazioni e dell'immagine IOS

Tutti i cambiamenti alle configurazione (running e startup) e all'immagine IOS del router vengono identificati: una possibile attività dell'attaccante dall'interno del router non passerà nascosta.

Gestione configurazione (CISCO-CONFIG-MAN-MIB) :

- ccmHistoryRunningLastChanged: “The value of sysUpTime when the running configuration was last changed.”;
- ccmHistoryRunningLastSaved: “The value of sysUpTime when the running configuration was last saved (written)”;

- ccmHistoryStartupLastChanged: “The value of sysUpTime when the startup configuration was last written to. In general this is the default configuration used when cold starting the system. It may have been changed by a save of the running configuration or by a copy from elsewhere”;
- ccmHistoryEventTable: “A table of configuration events on this router."

Gestione immagine IOS (CISCO-IMAGE-MIB):

- ciscoImageTable: "A table provides content information describing the executing IOS image."

7.3.10 Attività di backup della configurazione e dell'immagine IOS

RIDS mantiene un repository aggiornato dalla running-config e dell'immagine del sistema operativo (CISCO-IMAGE-UPGRADE-MIB e CISCO-CONFIG-COPY-MIB). Gestisce i cambiamenti ed è possibile fare un recovery a qualsiasi versione (stile CVS).

7.3.11 Fault hardware e software

Alcuni fault hardware e software sono: il malfunzionamento delle interfacce di rete (perdita del link), il surriscaldamento del router, la diminuzione della velocità delle ventole, il riavvio non volontario del router, l'uso intenso del processore e della memoria.

Tensione, temperatura e velocità delle ventole: CISCO-ENVMON-MIB:

- ciscoEnvMonVoltageStatusTable: "The table of voltage status maintained by the environmental monitor.";
- ciscoEnvMonTemperatureStatusTable: ”The table of ambient temperature status maintained by the environmental monitor.";
- ciscoEnvMonFanStatusTable: "The table of fan status maintained by the environmental monitor.";
- ciscoEnvMonSupplyStatusTable: "The table of power supply status maintained by the environmental monitor card."

Utilizzo della memoria: CISCO-MEMORY-POOL-MIB:

- ciscoMemoryPoolTable: "A table of memory pool monitoring entries.";
- ciscoMemoryPoolUtilizationTable: "A table of memory pool utilization entries. Each of the objects provides a general idea of how much of the memory pool has been used over a given period of time. It is determined as a weighted decaying average."

Carico del processore: CISCO-PROCESS-MIB:

- cpmCPUTotalTable: “A table of overall CPU statistics”. Contiene le statistiche dell'utilizzo di ogni processore negli ultimi cinque secondi, nell'ultimo minuto e negli ultimi cinque minuti.

Processi in esecuzione: CISCO-PROCESS-MIB

Controllando in tempo reale l'occupazione di risorse di ogni processo, RIDS interviene in modo mirato qualora le condizioni generali delle risorse siano critiche.

- cpmProcessTable: “A table of generic information on all active processes on this device”;
- cpmProcessExtRevTable: “Informazioni dettagliate sull'utilizzo delle risorse memoria/cpu e sui tempi”

La MIB OLD-CISCO-SYS contiene una serie di OID simili, tra cui molti deprecati, ma compatibili con la versione 1 di snmp.

7.4 **Risultati ottenuti**

Come ho precisato qualche pagina fa, RIDS è un codice d'esempio che implementa solamente alcuni dei controlli definiti del capitolo precedente, ma che sono sufficienti per dimostrare l'efficacia dell'approccio context-based nel campo degli Intrusion Detection System.

L'uso del programma è molto semplice:

```
embyte@portatile:~/lavoro/rids> ./rids
ERROR: Missing target and community
Usage: ./rids [options] target community
options:
```

```
        -l      load rids.conf
        -s      save rids.conf
        -t      calculate traffic rates
        -h      show this help
```

Un file di configurazione è creato alla prima esecuzione (nella fase di tuning): contiene indicazioni relative al target-system, ad alcuni parametri SNMP e al numero e tipo di interfacce monitorate.

```
target=172.16.0.1
community=stage
iface_number=4
monitored_iface_number=2
monitored_iface_name=FastEthernet0/0
monitored_iface_id=1
monitored_iface_name=FastEthernet0/1
monitored_iface_id=2
```

L'output generato durante l'inizializzazione di RIDS (con debug abilitato[1]) mostra in dettaglio le query SNMP al router e le relative risposte. Si distinguono la fase di identificazione del tipo di router, delle interfacce presenti, della realizzazione della configurazione dell'IDS e dell'acquisizione dei valori.

```
Monitoring 172.16.0.1 (community: stage)
DEBUG: ---- Quering router
DEBUG: Adding sysDescr.0 to get payload
************************************************
Working on:
Cisco Internetwork Operating System Software
IOS (tm) C2600 Software (C2600-IO3-M), Version 12.3(1), RELEASE
SOFTWARE (fc3)
Copyright (c) 1986-2003 by cisco Systems, Inc.
Compiled Thu 15-May-03 13:13 by dchihÉ
************************************************
DEBUG: ---- Quering router
DEBUG: Adding ifNumber.0 to get payload
Discovered 4 interfaces
Requesting more info for interfaces..
DEBUG: ---- Quering router
DEBUG: Adding ifDescr.1 to get payload
DEBUG: Adding ifAdminStatus.1 to get payload
DEBUG: Adding ifDescr.2 to get payload
```

1E' possibile disabilitare la modalità di debug andando a commentare la riga #define DEBUG in rids.h

```
DEBUG: Adding ifAdminStatus.2 to get payload
DEBUG: Adding ifDescr.3 to get payload
DEBUG: Adding ifAdminStatus.3 to get payload
DEBUG: Adding ifDescr.4 to get payload
DEBUG: Adding ifAdminStatus.4 to get payload
DEBUG: Adding iface FastEthernet0/0 (number 1) like important
DEBUG: Adding iface 1 like administrativement up
DEBUG: Adding iface FastEthernet0/1 (number 2) like important
DEBUG: Adding iface 2 like administrativement up
DEBUG: Adding iface 3 like administrativement up
DEBUG: Adding iface 4 like administrativement up
Discovered FastEthernet0/0: It's up and important. Signed to be
monitored
DEBUG: monitored_iface[0].name=FastEthernet0/0
DEBUG: monitored_iface[0].id=1
Discovered FastEthernet0/1: It's up and important. Signed to be
monitored
DEBUG: monitored_iface[1].name=FastEthernet0/1
DEBUG: monitored_iface[1].id=2
Null0 seems not important but administrativement up
do you want to monitor this one too? (y/n/Y/N): N

DEBUG: ---- Quering router
DEBUG: Adding ifAdminStatus.1 to get payload
DEBUG: Adding ifOperStatus.1 to get payload
DEBUG: Adding ifLastChange.1 to get payload
DEBUG: Adding ifInOctets.1 to get payload
DEBUG: Adding ifInDiscards.1 to get payload
DEBUG: Adding ifInErrors.1 to get payload
DEBUG: Adding ifInUnknownProtos.1 to get payload
DEBUG: Adding ifLinkUpDownTrapEnable.1 to get payload
DEBUG: Adding ifPromiscuousMode.1 to get payload
DEBUG: Adding ifAdminStatus.2 to get payload
DEBUG: Adding ifOperStatus.2 to get payload
DEBUG: Adding ifLastChange.2 to get payload
DEBUG: Adding ifInOctets.2 to get payload
DEBUG: Adding ifInDiscards.2 to get payload
DEBUG: Adding ifInErrors.2 to get payload
DEBUG: Adding ifInUnknownProtos.2 to get payload
DEBUG: Adding ifLinkUpDownTrapEnable.2 to get payload
DEBUG: Adding ifPromiscuousMode.2 to get payload
DEBUG: ---- INTERFACE FastEthernet0/0
DEBUG: Administrative state (IF-MIB::ifAdminStatus.1): 1
DEBUG: Operational state (IF-MIB::ifOperStatus.1): 1
DEBUG: LastChange time (IF-MIB::ifLastChange.1): 0:25:06.96
DEBUG: Inbound iface traffic (IF-MIB::ifInOctets.1): 1898698
byte
DEBUG: Inbound iface discarded packets (IF-MIB::ifInDiscards.1):
0
DEBUG: Inbound iface errored packets (IF-MIB::ifInErrors.1): 0
DEBUG: Inbound iface unknown protocol packets (IF-
MIB::ifInUnknownProtos.1): 0
DEBUG: Link UpDown Trap (IF-MIB::ifLinkUpDownTrapEnable.1): 1
```

```
DEBUG: Promiscuous mode (IF-MIB::ifPromiscuousMode.1): 2
DEBUG: ---- INTERFACE FastEthernet0/1
DEBUG: Administrative state (IF-MIB::ifAdminStatus.2): 1
DEBUG: Operational state (IF-MIB::ifOperStatus.2): 1
DEBUG: LastChange time (IF-MIB::ifLastChange.2): 0:00:13.34
DEBUG: Inbound iface traffic (IF-MIB::ifInOctets.2): 16794917
byte
DEBUG: Inbound iface discarded packets (IF-MIB::ifInDiscards.2):
0
DEBUG: Inbound iface errored packets (IF-MIB::ifInErrors.2): 0
DEBUG: Inbound iface unknown protocol packets (IF-
MIB::ifInUnknownProtos.2): 0
DEBUG: Link UpDown Trap (IF-MIB::ifLinkUpDownTrapEnable.2): 1
DEBUG: Promiscuous mode (IF-MIB::ifPromiscuousMode.2): 2
DEBUG: ---- Quering router
DEBUG: Adding ipInHdrErrors.0 to get payload
DEBUG: Adding ipInUnknownProtos.0 to get payload
DEBUG: Adding ipInDiscards.0 to get payload
DEBUG: -------------------- IP VALUES -------------------
DEBUG: Inbound ip errored packets (IP-MIB::ipInHdrErrors.0): 3
DEBUG: inbound ip unknown protocol packets (IP-
MIB::ipInUnknownProtos.0): 0
DEBUG: Inbound ip discarded packets (IP-MIB::ipInDiscards.0): 0
DEBUG: ---- Quering router
DEBUG: Adding tcpActiveOpens.0 to get payload
DEBUG: Adding tcpPassiveOpens.0 to get payload
DEBUG: Adding tcpAttemptFails.0 to get payload
DEBUG: Adding tcpInErrs.0 to get payload
DEBUG: Adding tcpOutRsts.0 to get payload
DEBUG: Adding tcpCurrEstab.0 to get payload
DEBUG: -------------------- TCP VALUES -------------------
DEBUG: Active opens (CLOSED->SYN-SENT) (TCP-
MIB::tcpActiveOpens.0): 0
DEBUG: Passive opens (LISTEN->SYN-RCVD) (TCP-
MIB::tcpPassiveOpens.0): 5
DEBUG: Failed connections (SYN-SENT,SYN-RVCD->CLOSED + SYN-RCVD-
>LISTEN) (TCP-MIB::tcpAttemptFails.0): 0
DEBUG: Inbound tcp errored packets (TCP-MIB::tcpInErrs.0): 0
DEBUG: Outbound RST packets (TCP-MIB::tcpOutRsts.0): 0
DEBUG: Established connection (ESTABLISHED, CLOSE-WAIT) (TCP-
MIB::tcpCurrEstab.0): 2
DEBUG: ---- Quering router
DEBUG: Adding udpInErrors.0 to get payload
DEBUG: Adding udpNoPorts.0 to get payload
DEBUG: -------------------- UDP VALUES -------------------
DEBUG: Inbound udp errored datagrams (UDP-MIB::udpInErrors.0): 0
DEBUG: Inbound udp datagrams to closed port (UDP-
MIB::udpNoPorts.0): 217
DEBUG: ---- Quering router
DEBUG: Adding icmpInErrors.0 to get payload
DEBUG: Adding icmpInEchos.0 to get payload
DEBUG: -------------------- ICMP VALUES -------------------
DEBUG: Inbound icmp errored messages (IP-MIB::icmpInErrors.0): 0
```

```
DEBUG: Inbound icmp echo-request received (IP-
MIB::icmpInEchos.0): 5
DEBUG: ---- Quering router
DEBUG: Adding snmpEnableAuthenTraps.0 to get payload
DEBUG: Adding snmpInBadVersions.0 to get payload
DEBUG: Adding snmpInBadCommunityNames.0 to get payload
DEBUG: Adding snmpInBadCommunityUses.0 to get payload
DEBUG: Adding snmpInASNParseErrs.0 to get payload
DEBUG: -------------------- SNMP VALUES -------------------
DEBUG: Authentication trap (SNMPv2-
MIB::snmpEnableAuthenTraps.0): 1
DEBUG: Inbound snmp bad version messages (SNMPv2-
MIB::snmpInBadVersions.0): 0
DEBUG: Inbound snmp bad community name message (SNMPv2-
MIB::snmpInBadCommunityNames.0): 0
DEBUG: Inbound snmp bad community uses message (SNMPv2-
MIB::snmpInBadCommunityUses.0): 0
DEBUG: ASN.1, BER errors by snmp entity (SNMPv2-
MIB::snmpInASNParseErrs.0): 0
Filling constant values
DEBUG: ---- Quering router
DEBUG: Adding ifSpeed.1 to get payload
DEBUG: Adding ifPhysAddress.1 to get payload
DEBUG: Adding ifSpeed.2 to get payload
DEBUG: Adding ifPhysAddress.2 to get payload
DEBUG: ------------------ CONST VALUES --------------------
DEBUG: Iface FastEthernet0/0 has ifSpeed set to 10000000
DEBUG: Iface FastEthernet0/0 has physical address equal to
STRING: 0:6:d7:b:ad:20
DEBUG: Iface FastEthernet0/1 has ifSpeed set to 100000000
DEBUG: Iface FastEthernet0/1 has physical address equal to
STRING: 0:6:d7:b:ad:21
Initialized completed
```

I test sono stati eseguiti simulando comportamenti anomali e intrusivi nei confronti del router e osservando l'output di RIDS. I risultati ottenuti sono elencati di seguito. La console del router (IP 172.16.0.1) è individuabile dal prompt Router(…)#, quella dell'IDS dal cancelletto. Gli allarmi del Router-IDS sono evidenziati in grassetto.

a. Passaggio dello stato operativo dell'interfaccia FastEthernet0/0 da UP a DOWN (perdita del link):

```
DEBUG: Administrative state (IF-MIB::ifAdminStatus.1): 1
DEBUG: Operational state (IF-MIB::ifOperStatus.1): 2
```

ALERT: FastEthernet0/0: Operational status is wrong (is 2, should be 1)

ALERT: FastEthernet0/0 has been modified in 2:03:36.96

b. Disabilitazione dell'interfaccia FastEthernet0/0 dalla shell del router. Lo stato amministrativo cambia da UP a DOWN:

```
ALERT: FastEthernet0/0: AdminStatus has changed from 1 to 2
ALERT: FastEthernet0/0 has been modified in 0:03:46.51
ALERT: FastEthernet0/0 has been modified in 0:04:06.82
```

c. Disabilitazione della trap SNMP sull'interfaccia FastEthernet0/0:

```
Router(config-if)#snmp trap ?
  link-status  Allow SNMP LINKUP and LINKDOWN traps
Router(config-if)#no snmp trap link-status

ALERT: FastEthernet0/0: Trap disabled
```

d. Portscan half-open su un range di porte 1-65000 del router:

```
# nmap -p 1-65000 -sS 172.16.0.1

DEBUG: Outbound RST packets (TCP-MIB::tcpOutRsts.0): 5559
ALERT: Transport layer (tcp): high tcpOutRsts value (187
packets/sec.)! Causes could be:
   1) Connection (connect()) against closed ports
   2) Halt-open portscan (-sS) to closed port
   3) Stealth FIN (-sF), Xmas Tree (-sX) or Null (-sN)
portscan modes
   4) Random tcp flood to closed port
```

e. Attacco syn-flood con RST da parte dell'attaccante. Questo attacco è una versione primita del syn-flood che mira solamente allo spreco di banda e non al consumo delle risorse di memoria del router (la connessione non rimane appena con socket in syn-rcvd):

```
# hping -i u100 -S -p 23 172.16.0.1
HPING 172.16.0.1 (eth0 172.16.0.1): S set, 40 headers + 0
data bytes
len=46 ip=172.16.0.1 ttl=255 id=0 sport=23 flags=SA
seq=1726 win=4128 rtt=3.6 ms
```

```
[cut]
--- 172.16.0.1 hping statistic ---
1727 packets tramitted, 1727 packets received, 0% packet
loss

DEBUG: Failed connections (SYN-SENT,SYN-RVCD->CLOSED + SYN-
RCVD->LISTEN) (TCP-MIB::tcpAttemptFails.0): 1500
ALERT: Possible syn-flood or half-open portscan (-sS)
againt open ports: high value of uncompleted 3 hand-shake:
101 > 100 (packets/sec.)
```

f. Attacco syn-flood:

```
# hping -S -p 23 -a 172.16.0.2 172.16.0.1 (.2 non esiste)
(ogni sec)

DEBUG: passive opens=12
DEBUG: attempts fails=0
ALERT: Discovered many appended connection with router
services: 12 (1 connecion/sec.)
```

g. Connessioni non legittime dal router all'esterno:

```
Router#telnet 172.16.0.50 21
Trying 172.16.0.50, 21 ... Open
220 ProFTPD 1.2.9 Server (ProFTPD Default Installation)
[portatile.madlab.org]

ALERT: Discovered 1 connection from router to outside
```

h. Se realizzo un numero elevato di connessioni dal router verso l'esterno:

```
ALERT: Possible tcp dos from router to outside: 13 > 10
(connection/sec.)
```

i. Consumo risorse di memoria sul router facendo molte connessioni verso il servizio di telnet:

```
# telnet 172.16.0.1 23 (5 volte)
```

```
ALERT: High established connection number: 5
```

j. Traffico alto:

```
DEBUG: FastEthernet0/0: Last medium traffic load is 5.84
KBps
DEBUG: Traffic-rate value correctly loaded: 221.20
ALERT: FastEthernet0/0: High traffic load: current traffic
(5.84) > estimated warning rate (0.27)
```

k. Portscan UDP:

```
# nmap -sU 172.16.0.1

DEBUG: Inbound udp datagrams to closed port (UDP-
MIB::udpNoPorts.0): 532
ALERT: Possible udp port-scanner: udp traffic (14) > udp
rate (10) (packets/sec.)
```

l. Attacco icmp-flood:

```
# ping -f 172.16.0.1

DEBUG: Inbound icmp echo-request received (IP-
MIB::icmpInEchos.0): 4841
ALERT: Possible icmp-dos: icmp traffic (402) > icmp rate
(100) (packets/sec.)
```

m. Disabilitazione della trap SNMP dell'autenticazione:

```
Router(config)#no snmp-server enable traps snmp
authentication

DEBUG: Authentication trap (SNMPv2-
MIB::snmpEnableAuthenTraps.0): 2
ALERT: Authentication failure traps disabled
```

n. Tentativo di accesso all'agent SNMP con community errata:

```
root@portatile:~# snmpget -v 2c -c community_sbagliata
172.16.0.1 sysDescr.0
Timeout: No Response from 172.16.0.1.

DEBUG: Inbound snmp bad community name message (SNMPv2-
MIB::snmpInBadCommunityNames.0): 6
```
ALERT: snmp: Found bad community name queries for 6 packets

8. RIDS

Questo capitolo raccoglie il sorgente di RIDS, rilasciato con licenza GPL. Per compilare è sufficiente usare il comando make.

Copyright (C) 2003 Marco Balduzzi
RIDS is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
You should have received a copy of the GNU General Public License along with this program in the file called “LICENSE”; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

```
----- CHANGELOG -----
0.1-public  2004-06-21

This is the first public realease.

This is a testing version and include many bugs!
It should not be used in production enviroment!

It's only a simple router-IDS coded during my
university stage like example of target-based model.
It's a very primitive tool and has many limitations.

More information in my thesis.

Tested on Linux 2.4 with Cisco 800 and 2600 series.
Net-SNMP library is required (http://www.net-snmp.org).

You can email-me with embyte@madlab.it.



----- LICENSE -----
GNU GENERAL PUBLIC LICENSE
                    Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.
                          59 Temple Place, Suite 330, Boston, MA  02111-
1307  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                        Preamble
```

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this

License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time. Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author
to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this. Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

```
Also add information on how to contact you by electronic and paper
mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type
`show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the
appropriate
parts of the General Public License.  Of course, the commands you use
may
be called something other than `show w' and `show c'; they could even
be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or
your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the
program
  `Gnomovision' (which makes passes at compilers) written by James
Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program
into
proprietary programs.  If your program is a subroutine library, you
may
consider it more useful to permit linking proprietary applications
with the
library.  If this is what you want to do, use the GNU Library General
Public License instead of this License.
```

```
----- Makefile -----
CFLAGS=-g -Wall -O2
LIBS = `net-snmp-config --libs`
OBJS = rids.o query.o monitor.o group.o \
       print_get.o debug.o memory.o \
       file_conf.o common.o check.o check_ext.o \
      t_rate.o

all:  $(OBJS)
      gcc $(CFLAGS) $(OBJS) -o rids $(LIBS)

.c.o:
      gcc $(CFLAGS) -c $< -o $@

clean:
      rm -fR *.o *~ rids

tarball:    clean
      mv ../rids-*_*.tar.gz ../rids_bkp.tar.gz
      cd .. && tar cfz rids-`date +%d_%m`.tar.gz rids
      ls -l ../rids_bkp.tar.gz ../rids-*_*.tar.gz
```

```
floppy:
      cp -f ../rids-*_*.tar.gz /mnt/floppy
      cp -f ../rids_bkp.tar.gz /mnt/floppy



----- rids.h -----
/*
 * RIDS - Router Intrusion Detection System
 *
 * (c) embyte <embyte@madlab.it>
 *
 * ICT Stage 2003
 *
 *
 * rids.h
 *
 * Global include file
 *
 */


#define _GNU_SOURCE           /* use glibc extensions */
#define DEBUG                   /* enable debug */

#define GROUP_NUMBER    6     /* check.c */
#define IDLE            2     /* idle time between two requests in
check.c (s) */
#define MIB_NUMBER      6     /* number of mib structs (iface, ip, tcp,
udp, icmp, snmp)
                         * elapsed time between same request is
IDLE*MIB_NUMBER */
#define MAX_OID_NAME    128   /* max oid length in char* form */
#define RATE_FILE "rids.t_rate"    /* configuration file for traffic
rates */
#define CONF_FILE "rids.conf" /* configuration file for ifaces */

#define TCP_AO_RATE     10    /* tcpActiveOpens rate */
#define TCP_PO_RATE     100   /* tcpPassiveOpens rate */
#define TCP_AF_RATE     100   /* tcpAttempsFails rate */
#define TCP_OR_RATE     100   /* tcpOutRsts rate */
#define TCP_ESTAB_RATE  10 //TODO   /* tcpCurrEstab */
#define PING_RATE       100   /* icmpInEchos rate */
#define UDP_NP_RATE     10    /* udpNoPorts rate */


#define CYAN   "\033[1;36m"   /* colors */
#define YELLOW "\033[1;33m"
#define RED    "\033[1;31m"
#define GRAY   "\033[2;37m"
#define NORM   "\033[0m"

#include <net-snmp/net-snmp-config.h>
#include <net-snmp/net-snmp-includes.h>
#include <unistd.h>
#include <errno.h>
/*
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdarg.h>
#include <errno.h>
*/
```

```
/* COMMON STRUCTS */

/* Requesting list which we copy to snmp request pdu */
typedef struct REQ_STRUCT
{
   char *name;
   struct REQ_STRUCT *next;
}
rids_req_list;

/* Vector of monitored interfaces */
struct IFACE_MONITORED
{
   u_char name[32];     /* ifName */
   u_short id;    /* ifNumber */
}
*monitored_iface;

/* Constant information for interfaces queried only one time */
struct IFACE_CONST_VALUES_STRUCT
{
   u_long ifSpeed;
   char ifPhysAddress[32];
}
*iface_const_values;


/*
 * Snmp MIB values (the old one)
 */
struct IFACE_VALUES_STRUCT
{
   u_short ifAdminStatus;     /* up(1), down(2), testing (3) */
   u_short ifOperStatus;      /* up(1), down(2), testing (3)
                         * unknow(4), dormant(5), notPresent(6),
lowerLayerDown(7) */
   u_long ifLastChange;       /* syntax: TimeTicks = time from which
OperStatus is up(1) */
   u_long ifInOctets;           /* rate */
   u_long ifInDiscards;
   u_long ifInErrors;
   u_long  ifInUnknownProtos;
   u_short ifLinkUpDownTrapEnable; /* enable(1), disable(2) */
   u_short ifPromiscuousMode; /* enable(1), disable(2) */
}
*iface_values;

struct IP_VALUES_STRUCT
{
   u_long ipInHdrErrors;
   u_long ipInUnknownProtos;
   u_long ipInDiscards;         /* pacchetti scartati per esaurimento
buffer d'ingresso */

   /* Not implemented yet */
   /* ipNetToMediaTable; */   /* arp -an */
}
ip_values;

struct TCP_VALUES_STRUCT
{
   u_long tcpActiveOpens;     /* rate */  /* connection from router to
outside */
   u_long tcpPassiveOpens;    /* rate */  /* 3 hand shake completed */
   u_long tcpAttemptFails;    /* rate */  /* syn to open port */
   u_long tcpInErrs;
   u_long tcpOutRsts;           /* rate */      /* connection resetted
```

```
: syn to closed port?*/
   u_long tcpCurrEstab;         /* rate */

   /* Not implemented yet */
   /* TcpConnTable; */        /* netstat -an */
}
tcp_values;

struct UDP_VALUES_STRUCT
{
   u_long udpInErrors;
   u_long udpNoPorts;   /* udp datagrams to closed ports */
}
udp_values;

struct ICMP_VALUES_STRUCT
{
   u_long icmpInErrors;
   u_long icmpInEchos;  /* rate */
}
icmp_values;

struct SNMP_VALUES_STRUCT
{
   u_short snmpEnableAuthenTraps;   /* enable(1), disabled(2) */
   u_long snmpInBadVersions;
   u_long snmpInBadCommunityNames;
   u_long snmpInBadCommunityUses;
   u_long snmpInASNParseErrs;

   //snmpOutGenErrs;
   //snmpOutBadValues;
}
snmp_values;

/* used by _group functions like parameter to store current values */
union GROUP
{
   struct IFACE_VALUES_STRUCT *iface_values;
   struct IP_VALUES_STRUCT     ip_values;
   struct TCP_VALUES_STRUCT    tcp_values;
   struct UDP_VALUES_STRUCT    udp_values;
   struct ICMP_VALUES_STRUCT   icmp_values;
   struct SNMP_VALUES_STRUCT   snmp_values;
};

/* GLOBAL VARIABLES */
typedef int turn;

char *sysDescr;
char *target;
char *community;
u_long ifMonitored;     /* number of monitored interfaces */
u_long ifNumber;        /* number of all interfaces */
char answer;            /* user input */
struct snmp_session *session; /* mono-thread session with router */

typedef struct
{
   long double a; /* 0-4 */
   long double b; /* 4-8 */
   long double c;       /* 8-12 */
   long double d;       /* 12-16 */
   long double e; /* 16-20 */
   long double f;       /* 20-24 */
}
time_slice;
```

```
/* PROTOS */

/* send a snmp request and read answer */
int query (struct snmp_session *session, rids_req_list *list, struct
snmp_pdu **answer);

/* requests list related functions */
rids_req_list * add_to_list (rids_req_list **in_list, const char
*format, ...);
void rids_free_list (rids_req_list **head, rids_req_list **list);
/* read iface id from oid in string format (oid.id) */
int oid_parse_iface (char *string);

/* main monitoring function */
int monitor ();
/* main checking function */
void check();
/* analize ifInOctets and build good offset alarm for traffic rate*/
int calculate_t_rate ();      /* t stand for traffic */
/* _group functions */
void netrequest_group (const char *type, union GROUP *values);
void save_group (const char *type, union GROUP *values);

/* debug and error functions */
void DEBUG_ME (const char *format, ...);
void alert (const char *format, ...);
void error (const char *format, ...);
void fatal (const char *format, ...);

/* secure malloc and remalloc */
void * xmalloc (size_t size);
void * xcalloc (size_t n, size_t eltsize);
void * xrealloc (void *p, size_t size) ;

/* prompts for a string or char */
void print_getline (char **in, const char *format, ...);
void print_getchar (char *in, const char *format, ...);

/* file protos */
void save_config ();
void load_config ();
long double load_t_rate (u_short iface_id, time_t present);

/* various rountines */
void dn (char **s);     /* delete \n */
char * uptimeString(u_long timeticks, char *buf);
char give_slice (time_t timep);
int rids_getline (char **lineptr, size_t *n, FILE *stream);
char * parse_confline (const char *s);
```

```
----- rids.c -----
/*
 * RIDS - Router Intrusion Detection System
 *
 * (c) embyte <embyte@madlab.it>
 *
 * ICT Stage 2003
 *
 *
 * rids.c
 *
```

```
 * main() file
 *
 */

#include "rids.h"

/* private functions */
static struct snmp_session * __init_sess
  (u_char *target, u_short version, u_char *community);
static int __discover_iface();
static void __show_banner();
static void __start_engine();
static void __show_help(char *);

/* pvt common vars */
struct variable_list *vars;
rids_req_list *list; /* requests list */
rids_req_list *head; /* head requests list */

struct snmp_pdu *response;

int main (int argc, char **argv)
{
   int arg;
   char **args;
   int i=0;

   struct flags_struct
     {
      char load_conf:1;
      char save_conf:1;
      char t_rate:1;
     }
   flags;

   /* Init variables */
   answer='\0';
   list=NULL;
   monitored_iface=NULL;
   memset (&flags, 0, sizeof (struct flags_struct));

   /* check args */
   while ((arg = getopt(argc, argv, ":lsth")) != EOF)
     {
      switch(arg)
        {
         case 'l':
           flags.load_conf=1;
           break;
         case 's':
           flags.save_conf=1;
           break;
         case 't':
           flags.t_rate=1;
           break;
         case 'h':
           __show_help(*argv);
           break;
         default:
           error ("Flag not recognized");
           __show_help(*argv);
           break;
        }
     }
   /* copy not flagged args */
   for (; optind<argc; optind++)
     (args[i++]=argv[optind]);
```

```
   if (flags.load_conf)
     {
           load_config();
     }
   else
     {
        /* read target and community */

      if (!i)
        {
           printf ("\n");
           error ("Missing target and community");
           __show_help(*argv);
        }
      if (i==1)
        {
           printf ("\n");
           error ("Missing community");
           __show_help(*argv);
        }

      asprintf(&target,args[0]);
      asprintf(&community, args[1]);

      /* for HOME
      asprintf(&target, "router");
      asprintf(&community, "stage"); */
     }

   /* print banner */
   __show_banner();

   printf ("Monitoring %s (community: %s)\n", target, community);

   /* Initialize session with router */
   if ((session = __init_sess(target, 2, community))==NULL)
     fatal ("InitializeSession() error. Quitting");

   /* Quering for router description */
   head=add_to_list(&list, "sysDescr.0");
   if (query(session, head, &response)<0)
     fatal ("Can't query sysDescr.0");

   rids_free_list(&head, &list);
   sysDescr=strdup(response->variables->val.string);
   snmp_free_pdu(response);

   printf ("************************************************\n");
   printf ("%sWorking on:%s\n", CYAN, NORM);
   printf ("%s\n", sysDescr);
   printf ("************************************************\n");

   /* if -t flag has been entered run traffic rate analyzer procedure
*/
   if (flags.t_rate)
     {
      if (calculate_t_rate()<0)
        return -1;
      else return 0;
     }

   /* If there is a config file start here! */
   if (flags.load_conf)
     __start_engine();

   /* ********************************* */
```

```
   /* Procedure to fill monitored_iface[] */
   /* ********************************** */
   else
     {
      if (__discover_iface()==-1)
        return -1;
     }

   /* snmp_close(session);    Leave it always open */

   if (flags.save_conf)
     save_config();

   __start_engine();

   return 0;
}

static struct snmp_session *
  __init_sess (u_char *target, u_short version, u_char *community)
{
   struct snmp_session prepare_session;
   struct snmp_session *session;

   init_snmp("RIDS");
   snmp_sess_init(&prepare_session);

   prepare_session.peername=strdup(target);    /* strdup calls malloc
and dupplicates string */
   switch (version)
     {
      case 1:
      prepare_session.version=SNMP_VERSION_1;
      break;
      case 2:
      prepare_session.version=SNMP_VERSION_2c;
      break;
    /* Not supported yet
    case 3:
      prepare_session.version=SNMP_VERSION_3;
      break;
    */
     }
   prepare_session.community=strdup(community);
   prepare_session.community_len=strlen(community);

   session = snmp_open(&prepare_session);

   if (session==NULL)
     {
      snmp_sess_perror("snmp_open", &prepare_session);
      return NULL;
     }

   return session;
}

static int __discover_iface()
{
   u_short i=0;
   u_short k=0;

   /*
    * Response value
    *
    * oid_value = vars->val
    * oid_name = snprint_objid()
```

```
     */
    union
      {
       u_long l;
       char *s; // allocated with strdup
      }
    oid_value;
    char oid_name[MAX_OID_NAME];

    int iface;
    struct IFACE_VALUTATE      /* used for finding what interface
monitor */
      {
       u_char name[32];/* set to iface name in router IF-MIB tree */
       u_short id; /* set to iface id in router IF-MIB tree */
       u_short imp;      /* set to 1 if it's important */
       u_short up; /* set to 1 if AdminStatus it's up (1) */
      }
    *valutate_iface;

    /**** END OF VARIABLES DECLARATION ****/

    /* Quering for interface number */
    head=add_to_list(&list, "ifNumber.0");
    if (query(session, head, &response)<0)
      fatal ("Can't query ifNumber.0");

    rids_free_list(&head, &list);

    if((ifNumber= *response->variables->val.integer)==0)
      fatal ("No interfaces! What??");

    printf ("%sDiscovered %lu interfaces%s\n", CYAN, ifNumber, NORM);
    snmp_free_pdu(response);

    /* Build iface[] vector for valutate what monitoring */
    valutate_iface = (struct IFACE_VALUTATE *) xcalloc (ifNumber,
sizeof (struct IFACE_VALUTATE));

    /* Quering for chose what interfaces monitoring */
    printf ("Requesting more info for interfaces..\n");

    /* build interfaces discover QUERY using this format:
     *
     * ifDescr.1
     * ifAdminStatus.1
     * ....
     * ifDescr.x
     * ifAdminStatus.x
     *
     * until x=ifNumber
     */
    for (i=1; i<=ifNumber; i++)
      {
         i==1 ? head=add_to_list(&list, "ifDescr.%d", i) :
add_to_list(&list, "ifDescr.%d", i);
       add_to_list(&list, "ifAdminStatus.%d", i);
      }

    if (query(session, head, &response)<0)
      fatal ("Can't query ifDescr. && ifAdminStatus.");

    rids_free_list(&head, &list);

    for (vars=response->variables; vars; vars=vars->next_variable)
      {
       /* Clear temp variables */
```

```
      oid_value.l=0;
      free (oid_value.s);
      memset (oid_name, 0, MAX_OID_NAME);

      /* parse oid vars->name */
      snprint_objid (oid_name, MAX_OID_NAME, vars->name, vars-
>name_length);

      /* parse iface number */
      if ((iface=oid_parse_iface (oid_name))==-1)
        {
           error ("Error: wrong oid in response: %s", oid_name);
           return -1;
        }

      /* set valutate_iface[] index k */
      k=iface-1;

        /* Add interface id to valutate_iface[] vector */
      valutate_iface[k].id=iface;

      /* parse vars->val value */
      switch (vars->type)
        {
         case ASN_OCTET_STR:
           oid_value.s=strdup(vars->val.string);
           break;
         case ASN_INTEGER:
           oid_value.l=*vars->val.integer;
           break;
        }

      /* parse oid_name parameter */
      if (strstr(oid_name, "ifDescr"))
        {
           /* Add interface description to valutate_iface[] vector */
             strncpy(valutate_iface[k].name, oid_value.s,
strlen(oid_value.s)*sizeof(char));

           if (strstr(oid_value.s, "Ethernet") ||
             strstr(oid_value.s, "BRI")) /* TODO: || altre interfacess
(adsl, fibra, hdsl...) */
             {
              DEBUG_ME ("Adding iface %s (number %d) like important",
oid_value.s, iface);
              valutate_iface[k].imp=1;
             }
        }

      else if (strstr(oid_name, "ifAdminStatus"))
        {
           if (oid_value.l==1) /* 1=state up */
             {
              DEBUG_ME ("Adding iface %d like administrativement up",
iface);
              valutate_iface[k].up=1;
             }
        }

      else /* error */
        {
           error ("Error: wrong oid in response: %s", oid_name);
           return -1;
        }

     }
```

```
   snmp_free_pdu(response);

   k=0;     /* counter for monitored_iface[] realloc() */

   for (i=0; i<ifNumber; i++)
     {
      if (!valutate_iface[i].imp && !valutate_iface[i].up) continue;
      /* ! important and down */
      else if (valutate_iface[i].imp && valutate_iface[i].up)    /*
important and up -> add directly to monitored_iface[] */
        {
           printf ("%sDiscovered %s: It's up and important. Signed to
be monitored%s\n", CYAN, valutate_iface[i].name, NORM);
        }
      else
        {
           if (answer=='N')
             continue;
           else if (answer=='Y');
           else   /* ask input from user */
             {
              do
                {
                 if (valutate_iface[i].imp && !valutate_iface[i].up)
                   /* important but down */
                    printf ("%s seems important but down\n",
                 valutate_iface[i].name);
                 if (!valutate_iface[i].imp && valutate_iface[i].up)
                   /* ! important but up */
                    printf ("%s seems not important but
                 administrativement up\n", valutate_iface[i].name);
                        print_getchar(&answer, "do you want to monitor
                 this one too? (y/n/Y/N): ");
                }
              while (answer!='y' && answer!='n' && answer!='Y' &&
answer!='N');

              if (answer!='y' && answer!='Y')
                continue;
             }
        }

      /*
       * adding procedure
       *
       * copy from valuate_iface[] to monitored_iface[]
       */

      /* ask for new memory */
      monitored_iface = (struct IFACE_MONITORED *) xrealloc
(monitored_iface, (k+1)*sizeof(struct IFACE_MONITORED));

      /* copy .name and .number variables */
        bzero (monitored_iface[k].name, 32*sizeof(u_char));
      strncpy(monitored_iface[k].name,
            valutate_iface[i].name,
            strlen(valutate_iface[i].name)*sizeof(char));

      monitored_iface[k].id=valutate_iface[i].id;

      DEBUG_ME("monitored_iface[%d].name=%s", k,
monitored_iface[k].name);
        DEBUG_ME("monitored_iface[%d].id=%d", k,
monitored_iface[k].id);

      /* new element has been added */
      k++;
```

```
     }

   /* free temp valutate_iface */
   if (valutate_iface)
     free (valutate_iface);

   ifMonitored = k;

   return 0;
}

static void __show_banner()
{
   printf ("\n%s", CYAN);
   printf ("\tRIDS\n");
   printf ("\tCoded by embyte <embyte@madlab.it>\n");
   printf ("\tICT Stage 2003\n");
   printf ("%s\n", NORM);
}

static void __start_engine()
{
   if (monitor()<0)
     fatal ("Fatal error in monitor()!");
}

static void __show_help(char *argv)
{
   printf ("\n%sRIDS Version 0.1-public%s\n", RED, NORM);
   printf ("\n%sUsage: %s [options] target community\n\n", CYAN,
argv);
   printf ("options:\n");
   printf ("\t-l\tload rids.conf\n");
   printf ("\t-s\tsave rids.conf\n");
   printf ("\t-t\tcalculate traffic rates\n");
   printf ("\t-h\tshow this help%s\n\n", NORM);

   exit (0);
}




----- check.c -----
#include "check.h"

/* Main function for checking value correctness */
void check()
{
   /* j is turn index and should be:
    * 0 iface
    * 1 ip
    * 2 tcp
    * 3 udp
    * 4 icmp
    * 5 snmp */
   static turn j;

   union GROUP *curr;

   /* allocate */
   curr = (union GROUP *) xcalloc (1, sizeof (union GROUP));
   curr->iface_values = (struct IFACE_VALUES_STRUCT *) xcalloc
(ifMonitored, sizeof (struct IFACE_VALUES_STRUCT));

   switch (j)
     {
```

```
      case 0:
      check_iface(curr);
      save_group ("iface", curr);
      break;
      case 1:
      check_ip(curr);
      save_group ("ip", curr);
      break;
      case 2:
      check_tcp(curr);
      save_group ("tcp", curr);
      break;
      case 3:
      check_udp(curr);
      save_group ("udp", curr);
      break;
      case 4:
      check_icmp(curr);
      save_group ("icmp", curr);
      break;
      case 5:
      check_snmp(curr);
      save_group ("snmp", curr);
      break;
      default:
      fatal ("Turn index wrong");
      break;
     }

   if (curr)
      free (curr);

   if (j==GROUP_NUMBER-1) j=0;
   else j++;

}



----- check_ext.c -----
#include "check.h"

/* calculate current traffic load */
static long double __calc_t (u_long past_t, u_long present_t);

u_long delta;     /* current - old */
static u_short REQ_IDLE=IDLE*MIB_NUMBER; /* idle time between the same
mib group */

void check_iface(union GROUP *curr_group)
{
   struct IFACE_VALUES_STRUCT *curr;
   u_short i;
   u_char *if_name;

   long double curr_t;  /* traffic load in bytes/sec. */
   long double t_rate;        /* traffic rate for iface */
   long double max_t;         /* brandwhich */

   netrequest_group ("iface", curr_group);
   curr=(struct IFACE_VALUES_STRUCT *)curr_group->iface_values;

   /* begin */
   for (i=0; i<ifMonitored; i++)
     {
```

```
      if_name=monitored_iface[i].name;

      if (curr[i].ifAdminStatus!=iface_values[i].ifAdminStatus)
        alert ("%s: AdminStatus has changed from %d to %d",
             if_name, iface_values[i].ifAdminStatus,
curr[i].ifAdminStatus);

      if (curr[i].ifOperStatus!=curr[i].ifAdminStatus)
        alert("%s: Operational status is wrong (is %d, should be %d)",
            if_name, curr[i].ifOperStatus, curr[i].ifAdminStatus);

      if (curr[i].ifLastChange!=iface_values[i].ifLastChange)
        {
           char timebuf[32];

           uptimeString (curr[i].ifLastChange, timebuf);
           alert("%s has been modified in %s", if_name, timebuf);
        }

      curr_t = __calc_t(iface_values[i].ifInOctets,
curr[i].ifInOctets)/1000;
      DEBUG_ME ("%s: Last medium traffic load is %.2Lf KBps", if_name,
curr_t);
      t_rate = load_t_rate (monitored_iface[i].id, time(NULL))/1000;
      max_t = (long double) iface_const_values[i].ifSpeed/8000;  /*
bps->KBps */

      if (curr_t>(t_rate*1.2))      /* t_rate + 20% */
        alert ("%s: High traffic load: current traffic (%.2Lf) >
estimated warning rate (%.2Lf) KBps",
             if_name, curr_t, (t_rate*1.2));

      if ((delta=curr[i].ifInDiscards-iface_values[i].ifInDiscards))
        alert("%s: Found discarded packets: %ld", if_name, delta);

      if ((delta=curr[i].ifInErrors-iface_values[i].ifInErrors))
        alert("%s: Found wrong packets: %ld", if_name, delta);

      if ((delta=curr[i].ifInUnknownProtos-
iface_values[i].ifInUnknownProtos))
        alert("%s: Unrecognized protocol for %ld packets", if_name,
delta);

      if (curr[i].ifLinkUpDownTrapEnable!=1)
        alert("%s: Trap disabled", if_name);

      if (curr[i].ifPromiscuousMode!=2)
        alert("%s: Promiscuous mode enabled", if_name);
     }
}

void check_ip(union GROUP *curr_group)
{
   struct IP_VALUES_STRUCT *curr;

   netrequest_group ("ip", curr_group);
   curr=(struct IP_VALUES_STRUCT *)&curr_group->ip_values;

   if ((delta=curr->ipInDiscards-ip_values.ipInDiscards))
     alert("Session layer (ip): Found discarded packets: %ld", delta);

   if ((delta=curr->ipInHdrErrors-ip_values.ipInHdrErrors))
     alert("Session layer (ip): Found wrong header for %ld packets",
delta);

   if ((delta=curr->ipInUnknownProtos-ip_values.ipInUnknownProtos))
     alert("Session layer (ip): Unrecognized protocol for %ld
```

```
packets", delta);
}

void check_tcp(union GROUP *curr_group)
{
   struct TCP_VALUES_STRUCT *curr;

   netrequest_group ("tcp", curr_group);
   curr=(struct TCP_VALUES_STRUCT *)&curr_group->tcp_values;

   if ((delta=curr->tcpActiveOpens-tcp_values.tcpActiveOpens))
     alert("Discovered %lu connection from router to outside", delta);

   if (delta/REQ_IDLE>TCP_AO_RATE)
     alert("Possible tcp dos from router to outside: %lu > %d
(connection/sec.)",
         (u_long) delta/REQ_IDLE, TCP_AO_RATE);

   delta=(curr->tcpPassiveOpens-tcp_values.tcpPassiveOpens)-(curr-
>tcpAttemptFails-tcp_values.tcpAttemptFails);

   /*
   DEBUG_ME ("passive opens=%d\n", curr->tcpPassiveOpens-
tcp_values.tcpPassiveOpens);
   DEBUG_ME ("attempts fails=%d\n", curr->tcpAttemptFails-
tcp_values.tcpAttemptFails);
   delta = curr->tcpPassiveOpens-tcp_values.tcpPassiveOpens-curr-
>tcpAttemptFails+tcp_values.tcpAttemptFails;
   */

   if (delta/REQ_IDLE>TCP_PO_RATE)  /* calc connection/second */
     alert("Discovered many connection with router services: %lu > %d
(connection/sec.)", (u_long) delta/REQ_IDLE, TCP_PO_RATE);

   delta=curr->tcpAttemptFails-tcp_values.tcpAttemptFails;
   if (delta/REQ_IDLE>TCP_AF_RATE)
     alert("Possible syn-flood or half-open portscan (-sS) againt open
ports: \n \
          high value of uncompleted 3 hand-shake: %lu > %d
(packets/sec.)",
         (u_long) delta/REQ_IDLE, TCP_AF_RATE);

   if ((delta=curr->tcpInErrs-tcp_values.tcpInErrs))
     alert("Transport layer (tcp): Found wrong header for %ld
packets", delta);

   delta=curr->tcpOutRsts-tcp_values.tcpOutRsts;
   if (delta/REQ_IDLE>TCP_OR_RATE)
     alert("Transport layer (tcp): high tcpOutRsts value (%lu
packets/sec.)! Causes could be:\n \
            1) Connection (connect()) against closed ports\n \
          2) Halt-open portscan (-sS) to closed port\n \
          3) Stealth FIN (-sF), Xmas Tree (-sX) or Null (-sN) portscan
modes\n \
          4) Random tcp flood to closed port", (u_long)
delta/REQ_IDLE);

   delta=curr->tcpCurrEstab-tcp_values.tcpCurrEstab;
   if (delta/REQ_IDLE>TCP_ESTAB_RATE)
     alert("High established connection number: %lu", (u_long)
delta/REQ_IDLE);
}

void check_udp(union GROUP *curr_group)
{
   struct UDP_VALUES_STRUCT *curr;
```

```
   netrequest_group ("udp", curr_group);
   curr=(struct UDP_VALUES_STRUCT *)&curr_group->udp_values;

   if ((delta=curr->udpInErrors-udp_values.udpInErrors))
     alert("Session layer (udp): Found wrong header for %ld packets",
delta);

   delta=curr->udpNoPorts-udp_values.udpNoPorts;
   if (delta/REQ_IDLE>UDP_NP_RATE)
     alert("Possible udp port-scanner: udp traffic (%lu) > udp rate
(%d) (packets/sec.)",
         (u_long) delta/REQ_IDLE, UDP_NP_RATE);
}

void check_icmp(union GROUP *curr_group)
{
   struct ICMP_VALUES_STRUCT *curr;

   netrequest_group ("icmp", curr_group);
   curr=(struct ICMP_VALUES_STRUCT *)&curr_group->icmp_values;

   if ((delta=curr->icmpInErrors-icmp_values.icmpInErrors))
     alert("Transport layer (icmp): Found wrong header for %ld
packets", delta);

   delta=curr->icmpInEchos-icmp_values.icmpInEchos;
   if (delta/REQ_IDLE>PING_RATE)
     alert("Possible icmp-dos: icmp traffic (%lu) > icmp rate (%d)
(packets/sec.)",
         (u_long) delta/REQ_IDLE, PING_RATE);
}

void check_snmp(union GROUP *curr_group)
{
   struct SNMP_VALUES_STRUCT *curr;

   netrequest_group ("snmp", curr_group);
   curr=(struct SNMP_VALUES_STRUCT *)&curr_group->snmp_values;

   if (curr->snmpEnableAuthenTraps!=1)
     alert ("Authentication failure traps disabled");

   if ((delta=curr->snmpInBadVersions-snmp_values.snmpInBadVersions))
     alert ("snmp: Found bad version queries for %ld packets", delta);

   if ((delta=curr->snmpInBadCommunityNames-
snmp_values.snmpInBadCommunityNames))
     alert ("snmp: Found bad community name queries for %ld packets",
delta);

   if ((delta=curr->snmpInBadCommunityUses-
snmp_values.snmpInBadCommunityUses))
     alert ("snmp: Found illegal community use queries for %ld
packets", delta);

   if ((delta=curr->snmpInASNParseErrs-
snmp_values.snmpInASNParseErrs))
     alert ("snmp: Found ASN parse errors for %ld packets", delta);
}

/********************/
/* private routines */

/* return medium traffic on iface */
/* PS: we don't implement first call rate */
static long double __calc_t (u_long past_t, u_long present_t)
{
```

```
   static time_t past;
   time_t present;
   int latency;

   if (!past)
     {
      past=time(NULL);
      return 0;
     }

   present=time(NULL);
   latency=(int) present-past;
   past=present;  /* update */

   return ((long double)(present_t-past_t)/latency);
}
```

```
----- check.h -----
/*
 * RIDS - Router Intrusion Detection System
 *
 * (c) embyte <embyte@madlab.it>
 *
 * ICT Stage 2003
 *
 */

#include "rids.h"

/* Protos for internal check() functions */
void check_iface(union GROUP *curr_group);
void check_ip(union GROUP *curr_group);
void check_tcp(union GROUP *curr_group);
void check_udp(union GROUP *curr_group);
void check_icmp(union GROUP *curr_group);
void check_snmp(union GROUP *curr_group);
```

```
----- common.c -----
#include "rids.h"

/* delete \n */
void dn (char **s)
{
   char *p=*s;

   if (p[strlen(p)-1]=='\n')
     p[strlen(p)-1]='\0';

   *s=p;
}

/**
 * @internal
 * Converts timeticks to hours, minutes, seconds string.
 *
 * @param timeticks    The timeticks to convert.
 * @param buf          Buffer to write to, has to be at
 *                     least 64 Bytes large.
 *
 * @return The buffer.
 */
```

```
char * uptimeString(u_long timeticks, char *buf)
{
   int             centisecs, seconds, minutes, hours, days;

   if (netsnmp_ds_get_boolean(NETSNMP_DS_LIBRARY_ID,
NETSNMP_DS_LIB_NUMERIC_TIMETICKS))
     {
        sprintf(buf, "%lu", timeticks);
        return buf;
     }

   centisecs = timeticks % 100;
   timeticks /= 100;
   days = timeticks / (60 * 60 * 24);
   timeticks %= (60 * 60 * 24);

   hours = timeticks / (60 * 60);
   timeticks %= (60 * 60);

   minutes = timeticks / 60;
   seconds = timeticks % 60;

   if (netsnmp_ds_get_boolean(NETSNMP_DS_LIBRARY_ID,
NETSNMP_DS_LIB_QUICK_PRINT))
     sprintf(buf, "%d:%d:%02d:%02d.%02d",
             days, hours, minutes, seconds, centisecs);
   else
     {
        if (days == 0)
        {
           sprintf(buf, "%d:%02d:%02d.%02d",
                 hours, minutes, seconds, centisecs);
        }
      else if (days == 1)
        {
           sprintf(buf, "%d day, %d:%02d:%02d.%02d",
                 days, hours, minutes, seconds, centisecs);
        }
      else
        {
           sprintf(buf, "%d days, %d:%02d:%02d.%02d",
                 days, hours, minutes, seconds, centisecs);
        }
     }
   return buf;
}


/* return current slice ID */
char give_slice (time_t timep)
{
   struct tm *now;

   now = localtime(&timep);

   if (now->tm_hour<4)
     return 'a';
   else if (now->tm_hour<8)
     return 'b';
   else if (now->tm_hour<12)
     return 'c';
   else if (now->tm_hour<16)
     return 'd';
   else if (now->tm_hour<20)
     return 'e';
   else
     return 'f';
```

```
}

/* call getline and erase \n */
int rids_getline (char **lineptr, size_t *n, FILE *stream)
{
   int retval;

   retval = getline (lineptr, n, stream);
   dn(lineptr);

   return retval;
}

char * parse_confline (const char *s)
{
   return strchr(s,'=')+1;
}
```

```
----- debug.c -----
#include "rids.h"

/* DEBUG (gray color) */
void DEBUG_ME (const char *format, ...)
{
#ifdef DEBUG
   va_list ap;

   va_start (ap, format);
   printf   ("%sDEBUG: ", GRAY);
   vprintf  (format, ap);
   printf   ("%s\n", NORM);
   fflush   (stdout);
   va_end   (ap);
#endif
}

/* ALERTS (yellow color) */
void alert (const char *format, ...)
{

   va_list ap;

   va_start (ap, format);
   fprintf  (stderr, "%sALERT: ", YELLOW);
   vfprintf (stderr, format, ap);
   fprintf  (stderr, "%s\n", NORM);
   fflush   (stderr);
   va_end   (ap);
}

/* ERRORS (red color) */
void error (const char *format, ...)
{
   va_list ap;

   va_start (ap, format);
   fprintf  (stderr, "%sERROR: ", RED);
   vfprintf (stderr, format, ap);
   fprintf  (stderr, "%s\n", NORM);
   fflush   (stderr);
   va_end   (ap);
}
```

```
/* FATAL ERRORS (red color) */
void fatal (const char *format, ...)
{
   va_list ap;

   va_start (ap, format);
   fprintf  (stderr, "%sERROR: ", RED);
   vfprintf (stderr, format, ap);
   fprintf  (stderr, "%s\n", NORM);
   fflush   (stderr);
   va_end   (ap);

   exit (EXIT_FAILURE);
}



----- file_conf.c -----
#include "rids.h"

FILE *stream;
u_short k=0;

/* save conf file */
void save_config ()
{
   if ((stream=fopen(CONF_FILE, "w"))==NULL)
     fatal (strerror(errno));

   fprintf (stream, "target=%s\n", target);
   fprintf (stream, "community=%s\n", community);
   fprintf (stream, "iface_number=%lu\n", ifNumber);
   fprintf (stream, "monitored_iface_number=%lu\n", ifMonitored);

   for (k=0; k<ifMonitored; k++)
     {
      fprintf (stream, "monitored_iface_name=%s\n",
monitored_iface[k].name);
      fprintf (stream, "monitored_iface_id=%d\n",
monitored_iface[k].id);
     }

   fflush (stream);
   fclose (stream);

   DEBUG_ME ("Configuration file correctly saved");
}

/* load conf file */
void load_config ()
{
   char *buffer=NULL;
   char *value;
   size_t n;
   u_short cnt=0;

   DEBUG_ME ("Attempt to read configuration file");

   if ((stream=fopen(CONF_FILE, "r"))==NULL)
      fatal ("%s: %s. Try to run rids with -s flag please.",
CONF_FILE, strerror(errno));

   while (rids_getline (&buffer, &n, stream) != -1)
     {
      if (strchr (buffer, '='))
        {
```

```
            value=parse_confline(buffer);

            if (strstr (buffer, "target"))
              {
               target=strdup(value);
               cnt++;
              }
            else if (strstr (buffer, "community"))
              {
               community=strdup(value);
               cnt++;
              }
            else if (strstr (buffer, "monitored_iface_number"))
              {
               ifMonitored=atol(value);
               cnt++;

               /* malloc monitored_iface[] */
               if (monitored_iface==NULL)
                 monitored_iface = (struct IFACE_MONITORED *)
                   xcalloc (ifMonitored, sizeof (struct
IFACE_MONITORED));
              }
            else if (strstr (buffer, "iface_number"))
              {
               ifNumber=atol(value);
               cnt++;
              }

            else if (strstr (buffer, "monitored_iface_name"))
              {
               sprintf (monitored_iface[k].name, value);
               cnt++;
              }
            else if (strstr (buffer, "monitored_iface_id"))
              {
               monitored_iface[k].id=atoi(value);
               cnt++;
               k++;      /* next one */
              }
            else
              continue;  /* no keyword :( */

            DEBUG_ME ("Found keyword in line: %s", buffer);
          }
      }

   if (buffer)
     free (buffer);
   fclose (stream);

   if (!ifMonitored || cnt!=ifMonitored*2+4)
     fatal ("Configuration file is wrong. Abort.");

   printf ("Configuration file correctly loaded\n");
}
```

```
----- memory.c ----
#include "rids.h"

void * xmalloc (size_t size)
{
   register void *value = malloc (size);
   if (!value)
```

```
     fatal ("virtual memory exhausted");
   return value;
}

void * xcalloc (size_t n, size_t eltsize)
{
   register void *value = calloc (n, eltsize);
   if (!value)
     fatal ("virtual memory exhausted");
   return value;
}

void * xrealloc (void *p, size_t size)
{
   register void *value = realloc (p, size);
   if (!value)
     fatal ("virtual memory exhausted");
   return value;
}



----- group.c -----
#include "rids.h"

/* internal routines */
void netrequest_iface (struct IFACE_VALUES_STRUCT **container);
void netrequest_ip    (struct IP_VALUES_STRUCT *container);
void netrequest_tcp   (struct TCP_VALUES_STRUCT *container);
void netrequest_udp   (struct UDP_VALUES_STRUCT *container);
void netrequest_icmp  (struct ICMP_VALUES_STRUCT *container);
void netrequest_snmp  (struct SNMP_VALUES_STRUCT *container);

/*
 * Copy the specific _VALUES_STRUCT from union to global STRUCT
 * declared in rids.h
 */
void save_group (const char *type, union GROUP *values)
{
   if (!strcmp (type, "iface"))
     memcpy (iface_values, values->iface_values, ifMonitored * sizeof
(struct IFACE_VALUES_STRUCT));
   else if (!strcmp (type, "ip"))
     memcpy (&ip_values, &values->ip_values, sizeof (struct
IP_VALUES_STRUCT));
   else if (!strcmp (type, "tcp"))
     memcpy (&tcp_values, &values->tcp_values, sizeof (struct
TCP_VALUES_STRUCT));
   else if (!strcmp (type, "udp"))
     memcpy (&udp_values, &values->udp_values, sizeof (struct
UDP_VALUES_STRUCT));
   else if (!strcmp (type, "icmp"))
     memcpy (&icmp_values, &values->icmp_values, sizeof (struct
ICMP_VALUES_STRUCT));
   else if (!strcmp (type, "snmp"))
     memcpy (&snmp_values, &values->snmp_values, sizeof (struct
SNMP_VALUES_STRUCT));
   else
     fatal ("Unknow struct type %s for save_group()", type);
}

/*
 * Fill the specific _VALUES_STRUCT with oid value readen
 * from snmp response
 */
void netrequest_group (const char *type, union GROUP *values)
{
```

```
   if (!strcmp (type, "iface"))
     netrequest_iface (&values->iface_values);
   else if (!strcmp (type, "ip"))
     netrequest_ip (&values->ip_values);
   else if (!strcmp (type, "tcp"))
     netrequest_tcp (&values->tcp_values);
   else if (!strcmp (type, "udp"))
     netrequest_udp (&values->udp_values);
   else if (!strcmp (type, "icmp"))
     netrequest_icmp (&values->icmp_values);
   else if (!strcmp (type, "snmp"))
     netrequest_snmp (&values->snmp_values);
   else
     fatal ("Unknow struct type %s for netrequest_group()", type);
}

/*****************************/
/* BEGIN OF INTERNAL ROUTINES */

/* COMMON PRIVATE VARIABLES */
rids_req_list *head, *list;
struct snmp_pdu *response;
struct variable_list *vars;
char oid_name[MAX_OID_NAME];
u_short i;

static void netrequest_init();

void netrequest_iface (struct IFACE_VALUES_STRUCT **container)
{
   u_short id;
   char timebuf[32];

   struct IFACE_VALUES_STRUCT *ptr = *container;
   const char *names[]=
     {
      "ifAdminStatus.",
        "ifOperStatus.",
        "ifLastChange.",
        "ifInOctets.",
        "ifInDiscards.",
        "ifInErrors.",
        "ifInUnknownProtos.",
        "ifLinkUpDownTrapEnable.",
        "ifPromiscuousMode."
     };

   netrequest_init();

   for (i=0; i<ifMonitored; i++)
     {
      if (!i) head=add_to_list (&list, "%s%d", *names,
monitored_iface[i].id);
      else add_to_list (&list, "%s%d", *names, monitored_iface[i].id);
      add_to_list (&list, "%s%d", *(names+1), monitored_iface[i].id);
      add_to_list (&list, "%s%d", *(names+2), monitored_iface[i].id);
      add_to_list (&list, "%s%d", *(names+3), monitored_iface[i].id);
      add_to_list (&list, "%s%d", *(names+4), monitored_iface[i].id);
      add_to_list (&list, "%s%d", *(names+5), monitored_iface[i].id);
      add_to_list (&list, "%s%d", *(names+6), monitored_iface[i].id);
      add_to_list (&list, "%s%d", *(names+7), monitored_iface[i].id);
      add_to_list (&list, "%s%d", *(names+8), monitored_iface[i].id);
     }

   if (query (session, head, &response)<0)
     fatal ("Can't query router");
```

```
   rids_free_list (&head, &list);

   for (vars=response->variables, i=0; vars; vars=vars->next_variable)
     {
      memset (oid_name, 0, MAX_OID_NAME);
      snprint_objid (oid_name, MAX_OID_NAME, vars->name, vars-
>name_length);
      id = oid_parse_iface(oid_name);

      if (id!=monitored_iface[i].id)
        {
           error ("Malformed snmp answer (%s): iface id order is wrong
\
                  (should be %d, is %d). Can't continue with response
parsing",
                oid_name, monitored_iface[i].id, id);
           break;
        }

      if (strstr (oid_name, *names))
        {
           DEBUG_ME ("-------------------- INTERFACE %s --------------
------", monitored_iface[i].name);

           ptr[i].ifAdminStatus=*(u_short *)vars->val.integer;
           DEBUG_ME ("Administrative state (%s): %d", oid_name,
ptr[i].ifAdminStatus);
        }
      else if (strstr (oid_name, *(names+1)))
        {
           ptr[i].ifOperStatus=*(u_short *)vars->val.integer;
           DEBUG_ME ("Operational state (%s): %d", oid_name,
ptr[i].ifOperStatus);
        }
      else if (strstr (oid_name, *(names+2)))
        {
           ptr[i].ifLastChange=*(u_long *)vars->val.integer;
           memset (timebuf, 0, 32);
           uptimeString(*(u_long *) (vars->val.integer), timebuf);
           DEBUG_ME ("LastChange time (%s): %s", oid_name, timebuf);
        }
      else if (strstr (oid_name, *(names+3)))
        {
           ptr[i].ifInOctets=*(u_long *)vars->val.integer;
           DEBUG_ME ("Inbound iface traffic (%s): %lu byte", oid_name,
ptr[i].ifInOctets);
        }
      else if (strstr (oid_name, *(names+4)))
        {
           ptr[i].ifInDiscards=*(u_long *)vars->val.integer;
           DEBUG_ME ("Inbound iface discarded packets (%s): %lu",
oid_name, ptr[i].ifInDiscards);
        }
      else if (strstr (oid_name, *(names+5)))
        {
           ptr[i].ifInErrors=*(u_long *) (vars->val.integer);
           DEBUG_ME ("Inbound iface errored packets (%s): %lu",
oid_name, ptr[i].ifInErrors);
        }
      else if (strstr (oid_name, *(names+6)))
        {
           ptr[i].ifInUnknownProtos=*(u_long *)vars->val.integer;
           DEBUG_ME ("Inbound iface unknown protocol packets (%s):
%lu", oid_name, ptr[i].ifInUnknownProtos);
        }
      else if (strstr (oid_name, *(names+7)))
        {
```

```
            ptr[i].ifLinkUpDownTrapEnable=*(u_short *)vars-
>val.integer;
            DEBUG_ME ("Link UpDown Trap (%s): %d", oid_name,
ptr[i].ifLinkUpDownTrapEnable);
          }
        else if (strstr (oid_name, *(names+8)))
          {
            ptr[i].ifPromiscuousMode=*(u_short *)vars->val.integer;
            DEBUG_ME ("Promiscuous mode (%s): %d", oid_name,
ptr[i].ifPromiscuousMode);

            /* next interface */
            i++;
          }
        else
          {
            error ("Wrong oid in response: %s", oid_name);
            break;
          }
      }
    snmp_free_pdu (response);

    *container=ptr;   /* save results */
}

void netrequest_ip (struct IP_VALUES_STRUCT *container)
{
    const char *names[]=
      {
       "ipInHdrErrors.0",
         "ipInUnknownProtos.0",
         "ipInDiscards.0"
      };

    netrequest_init();

    head=add_to_list (&list, *names);
    add_to_list (&list, *(names+1));
    add_to_list (&list, *(names+2));

    if (query (session, head, &response)<0)
      fatal ("Can't query router");

    rids_free_list (&head, &list);

    DEBUG_ME ("-------------------- IP VALUES -------------------");

    for (vars=response->variables; vars; vars=vars->next_variable)
      {
       memset (oid_name, 0, MAX_OID_NAME);
       snprint_objid (oid_name, MAX_OID_NAME, vars->name, vars-
>name_length);

        if (strstr (oid_name, *names))
          {
            container->ipInHdrErrors=*(u_long *)vars->val.integer;
            DEBUG_ME ("Inbound ip errored packets (%s): %lu", oid_name,
container->ipInHdrErrors);
          }
        else if (strstr (oid_name, *(names+1)))
          {
            container->ipInUnknownProtos=*(u_long *)vars->val.integer;
            DEBUG_ME ("inbound ip unknown protocol packets (%s): %lu",
oid_name, container->ipInUnknownProtos);
          }
        else if (strstr (oid_name, *(names+2)))
          {
```

```
           container->ipInDiscards=*(u_long *)vars->val.integer;
           DEBUG_ME ("Inbound ip discarded packets (%s): %lu",
oid_name, container->ipInDiscards);
         }
       else
         {
           error ("Wrong oid in response: %s", oid_name);
           break;
         }
     }

   snmp_free_pdu (response);
}

void netrequest_tcp (struct TCP_VALUES_STRUCT *container)
{
   const char *names[]=
     {
      "tcpActiveOpens.0",
        "tcpPassiveOpens.0",
        "tcpAttemptFails.0",
        "tcpInErrs.0",
        "tcpOutRsts.0",
        "tcpCurrEstab.0"
     };

   netrequest_init();

   head=add_to_list (&list, *names);
   for (i=1; i<6; i++)
     add_to_list (&list, *(names+i));

   if (query (session, head, &response)<0)
     fatal ("Can't query router");

   rids_free_list (&head, &list);

   DEBUG_ME ("-------------------- TCP VALUES -------------------");

   for (vars=response->variables; vars; vars=vars->next_variable)
     {
      memset (oid_name, 0, MAX_OID_NAME);
      snprint_objid (oid_name, MAX_OID_NAME, vars->name, vars-
>name_length);

      if (strstr (oid_name, *names))
        {
           container->tcpActiveOpens=*(u_long *)vars->val.integer;
           DEBUG_ME ("Active opens (CLOSED->SYN-SENT) (%s): %lu",
oid_name, container->tcpActiveOpens);
         }
      else if (strstr (oid_name, *(names+1)))
        {
           container->tcpPassiveOpens=*(u_long *)vars->val.integer;
           DEBUG_ME ("Passive opens (LISTEN->SYN-RCVD) (%s): %lu",
oid_name, container->tcpPassiveOpens);
         }
      else if (strstr (oid_name, *(names+2)))
        {
           container->tcpAttemptFails=*(u_long *)vars->val.integer;
           DEBUG_ME ("Failed connections (SYN-SENT,SYN-RVCD->CLOSED +
SYN-RCVD->LISTEN) (%s): %lu",
                  oid_name, container->tcpAttemptFails);
        }
      else if (strstr (oid_name, *(names+3)))
        {
           container->tcpInErrs=*(u_long *)vars->val.integer;
```

```
            DEBUG_ME ("Inbound tcp errored packets (%s): %lu",
oid_name, container->tcpInErrs);
          }
        else if (strstr (oid_name, *(names+4)))
          {
            container->tcpOutRsts=*(u_long *)vars->val.integer;
            DEBUG_ME ("Outbound RST packets (%s): %lu", oid_name,
container->tcpOutRsts);
          }
        else if (strstr (oid_name, *(names+5)))
          {
            container->tcpCurrEstab=*(u_long *)vars->val.integer;
            DEBUG_ME ("Established connection (ESTABLISHED, CLOSE-WAIT)
(%s): %lu",
                      oid_name, container->tcpCurrEstab);
          }
        else
          {
            print_variable (vars->name, vars->name_length, vars);
            error ("Wrong oid in response: %s", oid_name);
            break;
          }
      }

    snmp_free_pdu (response);
}

void netrequest_udp (struct UDP_VALUES_STRUCT *container)
{
    const char *names[]=
      {
       "udpInErrors.0",
         "udpNoPorts.0"
      };

    netrequest_init();

    head=add_to_list (&list, *names);
    add_to_list (&list, *(names+1));

    if (query (session, head, &response)<0)
      fatal ("Can't query router");

    rids_free_list (&head, &list);

    DEBUG_ME ("-------------------- UDP VALUES -------------------");

    for (vars=response->variables; vars; vars=vars->next_variable)
      {
       memset (oid_name, 0, MAX_OID_NAME);
       snprint_objid (oid_name, MAX_OID_NAME, vars->name, vars-
>name_length);

        if (strstr (oid_name, *names))
          {
            container->udpInErrors=*(u_long *)vars->val.integer;
            DEBUG_ME ("Inbound udp errored datagrams (%s): %lu",
oid_name, container->udpInErrors);
          }
        else if (strstr (oid_name, *(names+1)))
          {
            container->udpNoPorts=*(u_long *)vars->val.integer;
            DEBUG_ME ("Inbound udp datagrams to closed port (%s): %lu",
oid_name, container->udpNoPorts);
          }
        else
          {
```

```
            error ("Wrong oid in response: %s", oid_name);
            break;
          }
      }

   snmp_free_pdu (response);
}

void netrequest_icmp (struct ICMP_VALUES_STRUCT *container)
{
   const char *names[] =
     {
      "icmpInErrors.0",
        "icmpInEchos.0"
     };

   netrequest_init();

   head=add_to_list (&list, *names);
   add_to_list (&list, *(names+1));

   if (query (session, head, &response)<0)
     fatal ("Can't query router");

   rids_free_list (&head, &list);

   DEBUG_ME ("-------------------- ICMP VALUES -------------------");

   for (vars=response->variables; vars; vars=vars->next_variable)
     {
      memset (oid_name, 0, MAX_OID_NAME);
      snprint_objid (oid_name, MAX_OID_NAME, vars->name, vars-
>name_length);

      if (strstr (oid_name, *names))
        {
           container->icmpInErrors=*(u_long *)vars->val.integer;
           DEBUG_ME ("Inbound icmp errored messages (%s): %lu",
oid_name, container->icmpInErrors);
        }
      else if (strstr (oid_name, *(names+1)))
        {
           container->icmpInEchos=*(u_long *)vars->val.integer;
           DEBUG_ME ("Inbound icmp echo-request received (%s): %lu",
oid_name, container->icmpInEchos);
        }
      else
        {
           error ("Wrong oid in response: %s", oid_name);
           break;
        }
     }

   snmp_free_pdu (response);
}

void netrequest_snmp (struct SNMP_VALUES_STRUCT *container)
{
   const char *names[]=
     {
      "snmpEnableAuthenTraps.0",
        "snmpInBadVersions.0",
        "snmpInBadCommunityNames.0",
        "snmpInBadCommunityUses.0",
        "snmpInASNParseErrs.0",
     };
```

```
   netrequest_init();

   head=add_to_list (&list, *names);
   for (i=1; i<5; i++)
     add_to_list (&list, *(names+i));

   if (query (session, head, &response)<0)
     fatal ("Can't query router");

   rids_free_list (&head, &list);

   DEBUG_ME ("-------------------- SNMP VALUES ------------------");

   for (vars=response->variables; vars; vars=vars->next_variable)
     {
      memset (oid_name, 0, MAX_OID_NAME);
      snprint_objid (oid_name, MAX_OID_NAME, vars->name, vars-
>name_length);

      if (strstr (oid_name, *names))
        {
           container->snmpEnableAuthenTraps=*(u_short *)vars-
>val.integer;
           DEBUG_ME ("Authentication trap (%s): %d", oid_name,
container->snmpEnableAuthenTraps);
        }
      else if (strstr (oid_name, *(names+1)))
        {
           container->snmpInBadVersions=*(u_long *)vars->val.integer;
           DEBUG_ME ("Inbound snmp bad version messages (%s): %lu",
oid_name, container->snmpInBadVersions);
        }
      else if (strstr (oid_name, *(names+2)))
        {
           container->snmpInBadCommunityNames=*(u_long *)vars-
>val.integer;
           DEBUG_ME ("Inbound snmp bad community name message (%s):
%lu", oid_name, container->snmpInBadCommunityNames);
        }
      else if (strstr (oid_name, *(names+3)))
        {
           container->snmpInBadCommunityUses=*(u_long *)vars-
>val.integer;
           DEBUG_ME ("Inbound snmp bad community uses message (%s):
%lu", oid_name, container->snmpInBadCommunityUses);
        }
      else if (strstr (oid_name, *(names+4)))
        {
           container->snmpInASNParseErrs=*(u_long *)vars->val.integer;
           DEBUG_ME ("ASN.1, BER errors by snmp entity (%s): %lu",
oid_name, container->snmpInASNParseErrs);
        }
      else
        {
           error ("Wrong oid in response: %s", oid_name);
           break;
        }

     }

   snmp_free_pdu (response);
}

static void netrequest_init()
{
   list = NULL;
   if (monitored_iface == NULL)
```

```
     fatal ("Struct monitored_iface not initialized yet!");
}



----- monitor.c -----
/* base del monitoraggio */
#include "rids.h"

/* private sub routine */
int netrequest_const_values();

int monitor ()
{
   union GROUP *curr;

   /* allocate current */
   curr = (union GROUP *) xcalloc (1, sizeof (union GROUP));
   curr->iface_values = (struct IFACE_VALUES_STRUCT *) xcalloc
(ifMonitored, sizeof (struct IFACE_VALUES_STRUCT));

   /* Allocate iface_values (we don't know at begin ifMonitored! */
   iface_values = (struct IFACE_VALUES_STRUCT *) xcalloc (ifMonitored,
sizeof (struct IFACE_VALUES_STRUCT));
   iface_const_values = (struct IFACE_CONST_VALUES_STRUCT *) xcalloc
(ifMonitored, sizeof(struct IFACE_CONST_VALUES_STRUCT));

   /* Fill all VALUES structs */
   printf ("Saving current values\n");

   netrequest_group ("iface", curr);
   save_group ("iface", curr);
   netrequest_group ("ip", curr);
   save_group ("ip", curr);
   netrequest_group ("tcp", curr);
   save_group ("tcp", curr);
   netrequest_group ("udp", curr);
   save_group ("udp", curr);
   netrequest_group ("icmp", curr);
   save_group ("icmp", curr);
   netrequest_group ("snmp", curr);
   save_group ("snmp", curr);

   /* Fill iface_const_values */
   printf ("Filling constant values\n");

   if (netrequest_const_values()<0)
     return -1;

   printf ("%sInitialized completed%s\n", CYAN, NORM);

   /* Go into daemon mode */
   for (;;)
     {
      printf (". ");
      fflush (stdout);

      sleep(IDLE);
      check();
     }

   fatal ("I'm crashed!");
}

int netrequest_const_values()
{
```

```
   u_short i;
   u_short id;
   rids_req_list *head, *list;
   struct snmp_pdu *response;
   struct variable_list *vars;

   char oid_name[MAX_OID_NAME];
   list=NULL;

   if (monitored_iface==NULL)
     fatal ("Struct monitored_iface not initialized yet!");

   head=add_to_list(&list, "ifSpeed.%d", monitored_iface[0].id);
   add_to_list(&list, "ifPhysAddress.%d", monitored_iface[0].id);

   for (i=1; i<ifMonitored; i++)
     {
      add_to_list(&list, "ifSpeed.%d", monitored_iface[i].id);
      add_to_list(&list, "ifPhysAddress.%d", monitored_iface[i].id);
     }

   if (query (session, head, &response)<0)
     {
      error ("query() error");
      return -1;
     }

   rids_free_list (&head, &list);

   DEBUG_ME ("-------------------- CONST VALUES --------------------
");

   for (vars=response->variables, i=0; vars; vars=vars->next_variable)
     {
      memset (oid_name, 0, MAX_OID_NAME);
      snprint_objid (oid_name, MAX_OID_NAME, vars->name, vars-
>name_length);
      id=oid_parse_iface(oid_name);

      if (id!=monitored_iface[i].id)
        {
           error ("Malformed snmp answer (%s): iface id order is wrong
\
                  (should be %d, is %d). Can't continue with response
parsing",
                oid_name, monitored_iface[i].id, id);
           return -1;
        }

      if (strstr(oid_name, "ifSpeed"))
        {
           iface_const_values[i].ifSpeed=*(u_long *) vars-
>val.integer;
           DEBUG_ME ("Iface %s has ifSpeed set to %lu",
monitored_iface[i].name, iface_const_values[i].ifSpeed);
        }
      else if (strstr(oid_name, "ifPhysAddress"))    /* ifPhsyAddress
*/
        {
           // TODO

           snprint_value (iface_const_values[i].ifPhysAddress, 32,
vars->name, vars->name_length, vars);
           DEBUG_ME ("Iface %s has physical address equal to %s",
monitored_iface[i].name, iface_const_values[i].ifPhysAddress);

           /* pass to next iface */
```

```
            i++;
          }
        else
          {
            error ("wrong oid in response: %s", oid_name);
            return -1;
          }
      }

   snmp_free_pdu (response);

   return 0;
}



----- print_get.c -----
/* $Id: print_get.c, v 0.0.1 2003/08/05 11:17:58 embyte Exp $
 *
 * This file contains:
 *
 * * void print_getline (char **in, const char *format, ...)
 *   Write formatted output to stdout like printf(3)
 *   Read a line from stdin deleting final \n
 *
 * * void print_getchar (char *in, const char *format, ...)
 *   Call print_getline() and return the first char verifing
 *   input correctness: *in is set to \0 on NULL input or
 *   multiple chars input (see example in main)
 *
 * * int main()
 *   Two usage examples
 *
 *
 * Copyright (c) 2003, Embyte <embyte@madlab.it>
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
are met:
 *
 * *  Redistributions of source code must retain the above copyright
notice,
 *    this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above
copyright
 *    notice, this list of conditions and the following disclaimer in
the
 *    documentation and/or other materials provided with the
distribution.
 *
 * *  Neither the name of the Networks Associates Technology, Inc nor
the
 *    names of its contributors may be used to endorse or promote
 *    products derived from this software without specific prior
written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
``AS
 * IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR
```

```
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 */

#include "rids.h"

void print_getline (char **in, const char *format, ...)
{
   va_list ap;
   unsigned short lenght=0;
   char *s;

   /* Write formatted output to stdout */
   va_start (ap, format);
   vprintf (format, ap);
   fflush (stdout);
   va_end (ap);

   for (s=*in;;)
     {
      /* ask for memory (another char) */
      s=xrealloc(s, (lenght+1)*sizeof(char));

      if ((*(s+lenght)=getchar())=='\n')
        {
           *(s+lenght)='\0';  /* clear enter */
           break;
        }

      lenght++;
      if (lenght==1024) /* stop here to prevent DoS */
        break;
     }
   *in=s;
}

void print_getchar (char *in, const char *format, ...)     /* return
\0 on "error" */
{
   char *s;
   s=NULL;  /* important for the 1st realloc() call */

   print_getline (&s, format);

   if (*s=='\0')
     *in='\0';
   else if (*(s+1)!='\0')
     *in='\0';
   else
     *in=*s;
}



----- query.c -----
```

```
#include "rids.h"

int query (struct snmp_session *session, rids_req_list *list, struct
snmp_pdu **answer)
{
   struct snmp_pdu *pdu, /* payload requesting */
     *response;          /* payload answering */
   oid oid[MAX_OID_LEN]; /* computated oid */
   size_t oidlen;
   u_short i;
   int retval;

   rids_req_list *p;

   pdu=snmp_pdu_create(SNMP_MSG_GET);

   DEBUG_ME ("-------------------- Quering router --------------------
");

   for (p=list, i=0; p; p=p->next)
     {
      oidlen=MAX_OID_LEN;

      DEBUG_ME ("Adding %s to get payload", p->name);

      /* computate oid */
      if (get_node(p->name, oid, &oidlen)<0)
        {
           snmp_sess_perror("get_node", session);
           return -1;
        }
      /* add it with null value */
      snmp_add_null_var(pdu, oid, oidlen);
      i++;
     }

   /* send snmp request */
   retval=snmp_synch_response(session, pdu, &response);
   if (retval==STAT_SUCCESS && response->errstat==SNMP_ERR_NOERROR)
      /* SUCCESS */
     {
      *answer=response;
      return 0;
     }
   else     /* FAILURE */
     {
      if (retval==STAT_SUCCESS)
        error ("Packet error: %s", snmp_errstring (response-
>errstat));
      else
        snmp_sess_perror("rids", session);

      return -1;
     }
}

/* Add a new node to 'rids_req_list *in_list' using 'const char
*format' like input for inlist->name
 *
 * Returned value: list head (pointer to the first element) if
*in_list was empty, else nothing
 *
 * SEE: rids_req_list declaration in rids.h
 *      rids_free_list()
 */
rids_req_list * add_to_list (rids_req_list **in_list, const char
*format, ...)
```

```
{
   rids_req_list *p;
   rids_req_list *list;
   u_short namelen=128;

   va_list ap;
   char buffer[128];

   /* convert const char *format for buffer */
   va_start (ap, format);
   vsprintf (buffer, format, ap);
   va_end(ap);

   /* copy address */
   list=*in_list;

   /* list is empty */
   if (!list)
     {
      /* malloc for the first list node */
      list=xmalloc(sizeof(rids_req_list));
      /* copy *name */
      list->name=xcalloc(namelen, sizeof(char));
      sprintf (list->name, buffer);
      list->next=NULL;

      /* update in_list */
      *in_list=list;

      /* return list head */
      return list;
     }
   else
     {
        /* malloc for a new node */
      p=xmalloc (sizeof (rids_req_list));
      /* copy *name */
      p->name=xcalloc(namelen, sizeof(char));
      sprintf (p->name, buffer);
      p->next=NULL;

      /* add p to list */
      list->next=p;
      list=p;

      /* update in_list */
      *in_list=list;

      /* nothing to return */
      return NULL;
     }
}

/*
 * Free 'rids_req_list *list' and put *head=NULL
 *
 * SEE: rids_free_list declaration in rids.h
 */
void rids_free_list (rids_req_list **head, rids_req_list **list)
{
   rids_req_list *p;
   rids_req_list *h;

   h=*head;

   if (head==NULL || list==NULL)
     return;
```

```
   if (h->next)
     {
      while (h)
        {
           p=h->next; /* make p pointer to next node */
           free(h->name);
           free(h);
           h=p;
        }
     }
   else
     {
      free(h->name);
      free(h);
     }

   *list=NULL;
}

/*
 * INPUT: oid string in form oid.iface
 * RETURN: iface or -1 for errors
 */

int oid_parse_iface (char *string)
{
   char *j;

   j=strchr(string, '.');
   if (j==NULL)
     return -1;

   return atoi(j+1);

}



----- t_rate.c -----
/* This file include routines to calculate, load and save traffic rate
for each iface */

#include "rids.h"

static u_short __synch_time();

u_short i;
FILE *stream;

int calculate_t_rate ()
{
   time_slice *t_rate;

   struct variable_list *vars;
   struct snmp_pdu *response;
   rids_req_list *head, *list=NULL;

   u_long *curr_t_rate;
   u_long *old_t_rate;

   u_short first_call;

   u_short hour;
   int init_hour;
   char *buffer=NULL;
```

```
   time_t day_counter;
   int durate;

   printf ("\n%sRunning traffic-rate analysis and calculation
procedure%s\n", CYAN, NORM);

   if (ifNumber<1)
     {
      error ("Invalid interface number. Forcing loading rids.conf.
Next time use -l flag too");
      load_config();
      if (ifNumber<1)
        error ("Can't continue. Prior run rids with -s flag to write a
configuration file");
     }
   DEBUG_ME ("Interface number: %lu", ifNumber);

   /* allocate for current traffic rate vector */
   curr_t_rate=(u_long *) xcalloc(ifNumber, sizeof(u_long));
   old_t_rate=(u_long *) xcalloc(ifNumber, sizeof(u_long));
   t_rate=(time_slice *) xcalloc(ifNumber, sizeof(time_slice));

   do
     {
      print_getline(&buffer, "Insert data collection durate (in days):
");
      durate=atoi(buffer);
     }
   while (durate<=0);

   /* prepare query */
   for (i=1; i<=ifNumber; i++)
     {
      if (i==1) head=add_to_list(&list,"ifInOctets.1");
      else add_to_list(&list, "ifInOctets.%d",i);
     }

   /* begin */
   init_hour=-1;
   first_call=1;
   day_counter=0;

   while (day_counter<durate)
     {
      printf (". ");
      fflush (stdout);

      hour=__synch_time();

      if (init_hour<0)
        {
           init_hour=(u_short)hour; /* first call */
        }
      else if (init_hour==hour)
        {
           DEBUG_ME ("Another day is passed..");
           day_counter++;     /* another day passed */
        }

      if (query(session, head, &response)<0)
        fatal ("Can't query ifInOctets for calculate_t_rate()
routine");

      for (vars=response->variables, i=0; vars; vars=vars-
>next_variable, i++)
        {
```

```
            if (first_call)
              old_t_rate[i]=*(u_long *)vars->val.integer;
            else
              curr_t_rate[i]=*(u_long *)vars->val.integer;

            DEBUG_ME ("iface %d, t_rate is %lu bytes", i, *(u_long
*)vars->val.integer);
          }

        if (first_call)
          {
            first_call=0;
          }
        else
          {
            switch (hour)
              {
             case 4:
               for (i=0; i<ifNumber;i++)
                 t_rate[i].a+=curr_t_rate[i]-old_t_rate[i];
               break;
             case 8:
               for (i=0; i<ifNumber;i++)
                 t_rate[i].b+=curr_t_rate[i]-old_t_rate[i];
               break;
             case 12:
               for (i=0; i<ifNumber;i++)
                 t_rate[i].c+=curr_t_rate[i]-old_t_rate[i];
               break;
             case 16:
               for (i=0; i<ifNumber;i++)
                 t_rate[i].d+=curr_t_rate[i]-old_t_rate[i];
               break;
             case 20:
               for (i=0; i<ifNumber;i++)
                 t_rate[i].e+=curr_t_rate[i]-old_t_rate[i];
               break;
             case 0:
               for (i=0; i<ifNumber;i++)
                 t_rate[i].f+=curr_t_rate[i]-old_t_rate[i];
               break;
              }

            /* save current values like old */
            memcpy (old_t_rate, curr_t_rate, ifNumber*sizeof(u_long));
          }

        snmp_free_pdu(response);
      }

   /* free all */
   rids_free_list(&head, &list);
   if (curr_t_rate) free (curr_t_rate);
   if (old_t_rate) free (old_t_rate);

   /* computate global means */

   // test version
   for (i=0; i<ifNumber; i++)
     {
      t_rate[i].a/=5*durate;
      t_rate[i].b/=5*durate;
      t_rate[i].c/=5*durate;
      t_rate[i].d/=5*durate;
      t_rate[i].e/=5*durate;
      t_rate[i].f/=5*durate;
     }
```

```
   /*
   for (i=0; i<ifNumber; i++)
     {
      t_rate[i].a/=60*60*4*durate;  // 60 sec * 60 min * 4 hour *
durate day
      t_rate[i].b/=60*60*4*durate;
      t_rate[i].c/=60*60*4*durate;
      t_rate[i].d/=60*60*4*durate;
      t_rate[i].e/=60*60*4*durate;
      t_rate[i].f/=60*60*4*durate;
     }
   */


   /* RATE_FILE syntax:
    * ifaceN_TBEGIN-TEND=VALUE
    * ex: iface4_a=5023.32
    */

   /* save results */
   if ((stream=fopen(RATE_FILE, "w"))==NULL)
     fatal (strerror(errno));

   for (i=0; i<ifNumber; i++)
     {
      fprintf (stream, "iface%d_a=%.2Lf\n", i+1, t_rate[i].a);
      fprintf (stream, "iface%d_b=%.2Lf\n", i+1, t_rate[i].b);
      fprintf (stream, "iface%d_c=%.2Lf\n", i+1, t_rate[i].c);
      fprintf (stream, "iface%d_d=%.2Lf\n", i+1, t_rate[i].d);
      fprintf (stream, "iface%d_e=%.2Lf\n", i+1, t_rate[i].e);
      fprintf (stream, "iface%d_f=%.2Lf\n", i+1, t_rate[i].f);
     }

   fflush (stream);
   fclose (stream);

   printf ("\nOk I've done. Traffic-rate file correctly saved\n");

   return 0;
}

/* RATE_FILE syntax:
 * ifaceN_TBEGIN-TEND=VALUE
 * ex: iface4_a=5023.32
 */

/* load offset rate file (only one value) */
long double load_t_rate (u_short iface_id, time_t present)
{
   long double retval=0;
   u_short found=0;
   char *buffer=NULL;
   char patter[20];
   size_t n;

   if ((stream=fopen(RATE_FILE, "r"))==NULL)
     fatal ("%s: %s. Please run rids with -t flag.", RATE_FILE,
strerror(errno));

   memset (patter, 0, 20);
   sprintf (patter, "iface%d_%c", iface_id, give_slice (present));

   while (rids_getline (&buffer, &n, stream) != -1)
     {
      if (strstr(buffer, patter))
        {
```

```
            retval=strtold(parse_confline (buffer), NULL);
            found=1;
          }
      }

   fclose (stream);

   if (buffer)
     free (buffer);

   if (!found)
     fatal ("Patter not found in %s configuration file: %s",
RATE_FILE, patter);

   DEBUG_ME ("Traffic-rate value correctly loaded: %.2Lf", retval);

   return retval;
}

/*********************
 * PRIVATE ROUTINES
 */

/*
static u_short __synch_time()
{
   time_t t;
   struct tm *tm;

   for (;;)
     {
      sleep (60); // sleep a minute

      t=time(NULL);
      tm=localtime(&t);

      if (tm->tm_min==0)      // at :00
        {
           if (tm->tm_hour==0)      // 00:00
             return 0;
           else if (tm->tm_hour==4) // 04:00
             return 4;
           else if (tm->tm_hour==8) // 08:00
             return 8;
           else if (tm->tm_hour==12)      // 12:00
             return 12;
           else if (tm->tm_hour==16)      // 16:00
             return 16;
           else if (tm->tm_hour==20)      // 20:00
             return 20;
        }
     }
}
*/

/* testing version */
static u_short __synch_time()
{
   static turn a;
   short retval=1;

   switch (a)
     {
      case 0:
      retval=0;
      break;
      case 1:
```

```
      retval=4;
      break;
      case 2:
      retval=8;
      break;
      case 3:
      retval=12;
      break;
      case 4:
      retval=16;
      break;
      case 5:
      retval=20;
      break;
     }

   if (a==5)
     a=0;
   else
     a++;

   sleep (5);

   return retval;
}
```

9. Bibliografia

[1] Anderson, J. P., Computer security threat monitoring and surveillance. Technical Report, James P. Anderson Co., Washington PA, 1980.

[3] Research in Intrusion Detection Systems: A Survey, S. Axelsson. TR 98-17 (revised in 1999), Chalmers University of Technology, 1999.

[Ale96] Aleph1. Smashing the stack for fun and profit. (Phrack 49), 1996. http://www.phrack.org

[Axe00] Axelsson, S. (2000c). Intrusion Detection Systems: A Taxonomy and Survey. Technical Report 99-15, Dept. of Computer Engineering, Chalmers University of Technology.

[4] LibPcap library, http://www.tcpdump.org/

[5] CVE Common Vulnerability and Exposures, http://www.cve.mitre.org/

[CISCO1] Cisco, Configuring the Catalyst Switched Port Analyzer (SPAN) Document ID: 10570 http://www.cisco.com/warp/public/473\41.html

[CISCO2] Cisco IOS® IDS, http://www.cisco.com/warp/public/cc/pd/iosw/prodlit/ccids_qa.htm

[2] Emilie Lundin and Erland Jonsson. Survey of research in the intrusion detection area. Technical report 02-04, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, January 2002.

[IBM] RZ 3366 (# 93412) 09/03/2001 Computer Science 103 pages Research Report Malicious- and Accidental-Fault Tolerance for Internet Applications

Towards a Taxonomy of Intrusion Detection Systems and Attacks D. Alessandri*, C. Cachin*, M. Dacier*, O. Deak*, K. Julisch*, B. Randell!, J. Riordan*, A. Tscharner*, A. Wespi*, C. Wüest* *IBM Research Zurich Research Laboratory 8803 Rüschlikon Switzerland ! University of Newcastle upon Tyne United Kingdom

[SF1] Sourcefire 2003, Snort™ 2.0 - Detection Revisited, http://www.sourcefire.com/technology/whitepapers.html#detection

[SF2] Sourcefire 2003, Snort™ 2.0 - High Performance Multi-Rule Inspection Engine, http://www.sourcefire.com/technology/whitepapers.html#perform

[SF3] Sourcefire 2003, Snort™ 2.0 - Protocol Flow Analyzer, http://www.sourcefire.com/technology/whitepapers.html#flow

[SF4] Sourcefire 2003, Snort™ 2.0 - Rule Optimizer, http://www.sourcefire.com/technology/whitepapers.html#ruleop

[Roe99] Martin Roesch. Snort: Lightweight intrusion detection for networks. In LISA '99 Conference, November 1999.

[SNORT] Snort, The Open Source Network Intrusion Detection System, http://www.snort.org/

[SI] snort_inline IPS, http://snort-inline.sourceforge.net/

[JV94] Harold S. Javitz and Alfonso Valdes. The NIDES statistical component: Description and justification. Technical report, SRI Computer Science Laboratory, Menlo Park, CA, March 1994. http://www.sdl.sri.com/projects/nides/index5.html

[DBS92] H. Debar, M. Becker, and D. Siboni. A neural network component for an intrusion detection system. In Proceeding of the IEEE Symposium on Research in Computer Security and Privacy, pages 240 – 250, Oakland,

CA, 1992.

[TW] TripWire host-based IDS, http://www.tripwire.org

[AI] Aide HID, http://www.cs.tut.fi/~rammer/aide.html

[SW] Swatch log file tool, http://swatch.sf.net

[LS] Logsurfer, http://www.cert.dfn.de/eng/logsurf/

[LW] Logwatch log analysis system, http://www.logwatch.org

[GR] GrSecurity, http://www.grsecurity.net

[KSTAT] Kernel Security Therapy Anti-Trolls (2.4.x version) v1.1-2, http://www.s0ftpj.org/tools/kstat24_v1.1-2.tgz

[AT] Adore Teso Rootkit, http://www.team-teso.net

[TL] Simon Edwards and Top Layer, Vulnerability of Network Intrusion Detection Systems: Realizing and Overcoming the Risks. May 1, 2002

[IPT] Iptables and netfilter, http://www.netfilter.org/

[RIDS] RIDS Router-IDS, Marco Balduzzi, http://www.madlab.it/codes/rids-0.1-public.tar.gz

[SNMP] A Simple Network Management Protocol (SNMP): http://www.ietf.org/rfc/rfc1157.txt

[OID] Cisco SNMP Object Navigator, http://www.cisco.com/pcgi-bin/Support/Mibbrowser/unity.pl

[BSD] BSD License, http://www.opensource.org/licenses/bsd-license.php

[NSNMP] Net-SNMP, http://www.net-snmp.org/

[IOS] Cisco IOS, http://www.cisco.com/warp/public/732/Tech/

[RM] RMON Remote Monitoring RFC, http://www.ietf.org/rfc/rfc1757.txt

[CNF] Cisco Net-Flow, http://www.cisco.com/warp/public/732/Tech/nmp/netflow/index.shtml

[IRPAS] CDP vulnerability, Phenoelit, http://www.phenoelit.de/irpas/docu.html#cdp

[HP] Hping, Salvatore Sanfilippo, http://www.hping.org/

[SYNF] SYN Flood DoS Attack Experiments, http://www.niksula.cs.hut.fi/~dforsber/synflood/result.html

[NMAP] Nmap portscanner, 1997-2004 by Fyodor, http://www.insecure.org/nmap/

[RIPPER] Ripper, Mydecay & Click, S.P.I.N.E. Reaseach group, http://www.spine-group.org/tools/ripper-0.1.4.tar.gz

[SPINE] S.P.I.N.E. Rearch group, http://www.spine-group.org

[ASPRD] Marco Balduzzi, Michele Marchetto e Valerio Genovese. Analisi di sicurezza dei protocolli di routing dinamico. SecurityDate 2004 Ancona, 29 Aprile. http://www.madlab.it/slides/secdate04.pdf

[GPL] Licenza Opensource GPL, www.gnu.org/copyleft/gpl.html

[CGI] Cgi probe, http://www.packetstormsecurity.com/groups/s0ftpj/cgiscan.c

10. Riferimenti

1) Related Work in Intrusion Detection, http://www.cs.fit.edu/~pkc/id/related/

2) Stefano Zanero. Tesi di laurea, 2002. Un sistema di intrusion detection basato su tecniche di apprendimento non supervisionato, http://www.elet.polimi.it/upload/zanero/papers/IDS-tesi.pdf

3) FOCUS-IDS Mailing-list, http://www.securityfocus.com/archive/96.